Sphinx Documentation

Release 1.3.1

Georg Brandl

September 22, 2015

Contents

1 Introduction **1**
- 1.1 Conversion from other systems . . . 1
- 1.2 Use with other systems . . . 2
- 1.3 Prerequisites . . . 2
- 1.4 Usage . . . 2

2 First Steps with Sphinx **3**
- 2.1 Install Sphinx . . . 3
- 2.2 Setting up the documentation sources . . . 3
- 2.3 Defining document structure . . . 3
- 2.4 Adding content . . . 4
- 2.5 Running the build . . . 5
- 2.6 Documenting objects . . . 5
- 2.7 Basic configuration . . . 6
- 2.8 Autodoc . . . 6
- 2.9 Intersphinx . . . 7
- 2.10 More topics to be covered . . . 7

3 Invocation of sphinx-quickstart **9**
- 3.1 Structure options . . . 9
- 3.2 Project basic options . . . 9
- 3.3 Extension options . . . 10
- 3.4 Makefile and Batchfile creation options . . . 10

4 Invocation of sphinx-build **11**
- 4.1 Makefile options . . . 13

5 Invocation of sphinx-apidoc **15**

6 reStructuredText Primer **17**
- 6.1 Paragraphs . . . 17
- 6.2 Inline markup . . . 17
- 6.3 Lists and Quote-like blocks . . . 18
- 6.4 Source Code . . . 19
- 6.5 Tables . . . 20
- 6.6 Hyperlinks . . . 20
- 6.7 Sections . . . 20
- 6.8 Explicit Markup . . . 21
- 6.9 Directives . . . 21

6.10 Images . . . 23
6.11 Footnotes . . . 24
6.12 Citations . . . 24
6.13 Substitutions . . . 24
6.14 Comments . . . 25
6.15 Source encoding . . . 25
6.16 Gotchas . . . 25

7 Sphinx Markup Constructs **27**
7.1 The TOC tree . . . 27
7.2 Paragraph-level markup . . . 30
7.3 Table-of-contents markup . . . 32
7.4 Glossary . . . 32
7.5 Grammar production displays . . . 33
7.6 Showing code examples . . . 33
7.7 Inline markup . . . 37
7.8 Miscellaneous markup . . . 42

8 Sphinx Domains **47**
8.1 What is a Domain? . . . 47
8.2 Basic Markup . . . 47
8.3 The Python Domain . . . 48
8.4 The C Domain . . . 53
8.5 The C++ Domain . . . 54
8.6 The Standard Domain . . . 58
8.7 The JavaScript Domain . . . 59
8.8 The reStructuredText domain . . . 60
8.9 More domains . . . 61

9 Available builders **63**
9.1 Serialization builder details . . . 67

10 The build configuration file **69**
10.1 General configuration . . . 69
10.2 Project information . . . 72
10.3 Options for internationalization . . . 74
10.4 Options for HTML output . . . 76
10.5 Options for Apple Help output . . . 81
10.6 Options for epub output . . . 83
10.7 Options for LaTeX output . . . 86
10.8 Options for text output . . . 89
10.9 Options for manual page output . . . 89
10.10 Options for Texinfo output . . . 90
10.11 Options for the linkcheck builder . . . 91
10.12 Options for the XML builder . . . 92

11 Internationalization **93**
11.1 Sphinx internationalization details . . . 94
11.2 Translating with sphinx-intl . . . 95
11.3 Using Transifex service for team translation . . . 96
11.4 Contributing to Sphinx reference translation . . . 97

12 HTML theming support **99**
12.1 Using a theme . . . 99
12.2 Builtin themes . . . 102

12.3 Creating themes . . . 105

13 Templating **107**
13.1 Do I need to use Sphinx's templates to produce HTML? . . . 107
13.2 Jinja/Sphinx Templating Primer . . . 107
13.3 Working with the builtin templates . . . 108

14 Sphinx Extensions **113**
14.1 Builtin Sphinx extensions . . . 113
14.2 Third-party extensions . . . 142

15 Developing extensions for Sphinx **143**
15.1 Extension metadata . . . 143
15.2 APIs used for writing extensions . . . 144

16 Sphinx Web Support **169**
16.1 Web Support Quick Start . . . 169
16.2 The WebSupport Class . . . 173
16.3 Search Adapters . . . 176
16.4 Storage Backends . . . 178

17 Sphinx FAQ **181**
17.1 How do I... . . . 181
17.2 Using Sphinx with... . . . 181
17.3 Epub info . . . 182
17.4 Texinfo info . . . 183

18 Glossary **185**

19 Sphinx Developer's Guide **187**
19.1 Bug Reports and Feature Requests . . . 187
19.2 Contributing to Sphinx . . . 187
19.3 Coding Guide . . . 190

20 Changes in Sphinx **193**
20.1 Release 1.4 (in development) . . . 193
20.2 Release 1.3.1 (released Mar 17, 2015) . . . 195
20.3 Release 1.3 (released Mar 10, 2015) . . . 195
20.4 Release 1.3b3 (released Feb 24, 2015) . . . 196
20.5 Release 1.3b2 (released Dec 5, 2014) . . . 197
20.6 Release 1.3b1 (released Oct 10, 2014) . . . 198
20.7 Release 1.2.3 (released Sep 1, 2014) . . . 202
20.8 Release 1.2.2 (released Mar 2, 2014) . . . 203
20.9 Release 1.2.1 (released Jan 19, 2014) . . . 203
20.10 Release 1.2 (released Dec 10, 2013) . . . 205
20.11 Release 1.2 beta3 (released Oct 3, 2013) . . . 206
20.12 Release 1.2 beta2 (released Sep 17, 2013) . . . 206
20.13 Release 1.2 beta1 (released Mar 31, 2013) . . . 208
20.14 Release 1.1.3 (Mar 10, 2012) . . . 212
20.15 Release 1.1.2 (Nov 1, 2011) – 1.1.1 is a silly version number anyway! . . . 213
20.16 Release 1.1.1 (Nov 1, 2011) . . . 213
20.17 Release 1.1 (Oct 9, 2011) . . . 213
20.18 Release 1.0.8 (Sep 23, 2011) . . . 215
20.19 Release 1.0.7 (Jan 15, 2011) . . . 216
20.20 Release 1.0.6 (Jan 04, 2011) . . . 217

20.21 Release 1.0.5 (Nov 12, 2010) . . . 217
20.22 Release 1.0.4 (Sep 17, 2010) . . . 218
20.23 Release 1.0.3 (Aug 23, 2010) . . . 218
20.24 Release 1.0.2 (Aug 14, 2010) . . . 218
20.25 Release 1.0.1 (Jul 27, 2010) . . . 219
20.26 Release 1.0 (Jul 23, 2010) . . . 219
20.27 Previous versions . . . 222

21 Projects using Sphinx **223**
21.1 Documentation using the default theme . . . 223
21.2 Documentation using a customized version of the default theme . . . 224
21.3 Documentation using the sphinxdoc theme . . . 225
21.4 Documentation using another builtin theme . . . 226
21.5 Documentation using a custom theme/integrated in a site . . . 226
21.6 Homepages and other non-documentation sites . . . 228
21.7 Books produced using Sphinx . . . 228
21.8 Thesis using Sphinx . . . 228

22 Sphinx authors **229**

Python Module Index **231**

CHAPTER 1

Introduction

This is the documentation for the Sphinx documentation builder. Sphinx is a tool that translates a set of reStructuredText[1] source files into various output formats, automatically producing cross-references, indices etc. That is, if you have a directory containing a bunch of reST-formatted documents (and possibly subdirectories of docs in there as well), Sphinx can generate a nicely-organized arrangement of HTML files (in some other directory) for easy browsing and navigation. But from the same source, it can also generate a LaTeX file that you can compile into a PDF version of the documents, or a PDF file directly using rst2pdf[2].

The focus is on hand-written documentation, rather than auto-generated API docs. Though there is support for that kind of docs as well (which is intended to be freely mixed with hand-written content), if you need pure API docs have a look at Epydoc[3], which also understands reST.

For a great "introduction" to writing docs in general – the whys and hows, see also Write the docs[4], written by Eric Holscher.

1.1 Conversion from other systems

This section is intended to collect helpful hints for those wanting to migrate to reStructuredText/Sphinx from other documentation systems.

- Gerard Flanagan has written a script to convert pure HTML to reST; it can be found at the Python Package Index[5].
- For converting the old Python docs to Sphinx, a converter was written which can be found at the Python SVN repository[6]. It contains generic code to convert Python-doc-style LaTeX markup to Sphinx reST.
- Marcin Wojdyr has written a script to convert Docbook to reST with Sphinx markup; it is at Google Code[7].
- Christophe de Vienne wrote a tool to convert from Open/LibreOffice documents to Sphinx: odt2sphinx[8].
- To convert different markups, Pandoc[9] is a very helpful tool.

[1] http://docutils.sf.net/rst.html
[2] http://rst2pdf.googlecode.com
[3] http://epydoc.sf.net/
[4] http://write-the-docs.readthedocs.org/
[5] https://pypi.python.org/pypi/html2rest
[6] http://svn.python.org/projects/doctools/converter
[7] http://code.google.com/p/db2rst/
[8] https://pypi.python.org/pypi/odt2sphinx/
[9] http://johnmacfarlane.net/pandoc/

1.2 Use with other systems

See the *pertinent section in the FAQ list*.

1.3 Prerequisites

Sphinx needs at least **Python 2.6** or **Python 3.3** to run, as well as the docutils[10] and Jinja2[11] libraries. Sphinx should work with docutils version 0.10 or some (not broken) SVN trunk snapshot. If you like to have source code highlighting support, you must also install the Pygments[12] library.

1.4 Usage

See First Steps with Sphinx for an introduction. It also contains links to more advanced sections in this manual for the topics it discusses.

[10] http://docutils.sf.net/
[11] http://jinja.pocoo.org/
[12] http://pygments.org/

CHAPTER 2

First Steps with Sphinx

This document is meant to give a tutorial-like overview of all common tasks while using Sphinx.

The green arrows designate "more info" links leading to advanced sections about the described task.

2.1 Install Sphinx

Install Sphinx, either from a distribution package or from PyPI[1] with

```
$ pip install Sphinx
```

2.2 Setting up the documentation sources

The root directory of a Sphinx collection of reStructuredText document sources is called the *source directory*. This directory also contains the Sphinx configuration file `conf.py`, where you can configure all aspects of how Sphinx reads your sources and builds your documentation. [2]

Sphinx comes with a script called **`sphinx-quickstart`** that sets up a source directory and creates a default `conf.py` with the most useful configuration values from a few questions it asks you. Just run

```
$ sphinx-quickstart
```

and answer its questions. (Be sure to say yes to the "autodoc" extension.)

There is also an automatic "API documentation" generator called **`sphinx-apidoc`**; see *Invocation of sphinx-apidoc* for details.

2.3 Defining document structure

Let's assume you've run **`sphinx-quickstart`**. It created a source directory with `conf.py` and a master document, `index.rst` (if you accepted the defaults). The main function of the *master document* is to serve as a welcome page, and to contain the root of the "table of contents tree" (or *toctree*). This is one of the main things that Sphinx adds to reStructuredText, a way to connect multiple files to a single hierarchy of documents.

[1] https://pypi.python.org/pypi/Sphinx

[2] This is the usual layout. However, `conf.py` can also live in another directory, the *configuration directory*. See *Invocation of sphinx-quickstart*.

reStructuredText directives

`toctree` is a reStructuredText *directive*, a very versatile piece of markup. Directives can have arguments, options and content.
Arguments are given directly after the double colon following the directive's name. Each directive decides whether it can have arguments, and how many.
Options are given after the arguments, in form of a "field list". The `maxdepth` is such an option for the `toctree` directive.
Content follows the options or arguments after a blank line. Each directive decides whether to allow content, and what to do with it.
A common gotcha with directives is that **the first line of the content must be indented to the same level as the options are**.

The toctree directive initially is empty, and looks like this:

```
.. toctree::
   :maxdepth: 2
```

You add documents listing them in the *content* of the directive:

```
.. toctree::
   :maxdepth: 2

   intro
   tutorial
   ...
```

This is exactly how the toctree for this documentation looks. The documents to include are given as *document name*s, which in short means that you leave off the file name extension and use slashes as directory separators.

Read more about *the toctree directive*.

You can now create the files you listed in the toctree and add content, and their section titles will be inserted (up to the "maxdepth" level) at the place where the toctree directive is placed. Also, Sphinx now knows about the order and hierarchy of your documents. (They may contain `toctree` directives themselves, which means you can create deeply nested hierarchies if necessary.)

2.4 Adding content

In Sphinx source files, you can use most features of standard reStructuredText. There are also several features added by Sphinx. For example, you can add cross-file references in a portable way (which works for all output types) using the *`ref`* role.

For an example, if you are viewing the HTML version you can look at the source for this document – use the "Show Source" link in the sidebar.

See *reStructuredText Primer* for a more in-depth introduction to reStructuredText and *Sphinx Markup Constructs* for a full list of markup added by Sphinx.

2.5 Running the build

Now that you have added some files and content, let's make a first build of the docs. A build is started with the **sphinx-build** program, called like this:

```
$ sphinx-build -b html sourcedir builddir
```

where *sourcedir* is the *source directory*, and *builddir* is the directory in which you want to place the built documentation. The `-b` option selects a builder; in this example Sphinx will build HTML files.

See *Invocation of sphinx-quickstart* for all options that **sphinx-build** supports.

However, **sphinx-quickstart** script creates a `Makefile` and a `make.bat` which make life even easier for you: with them you only need to run

```
$ make html
```

to build HTML docs in the build directory you chose. Execute `make` without an argument to see which targets are available.

How do I generate PDF documents?

`make latexpdf` runs the *LaTeX builder* and readily invokes the pdfTeX toolchain for you.

2.6 Documenting objects

One of Sphinx's main objectives is easy documentation of *objects* (in a very general sense) in any *domain*. A domain is a collection of object types that belong together, complete with markup to create and reference descriptions of these objects.

The most prominent domain is the Python domain. To e.g. document the Python built-in function `enumerate()`, you would add this to one of your source files:

```
.. py:function:: enumerate(sequence[, start=0])

   Return an iterator that yields tuples of an index and an item of the
   *sequence*. (And so on.)
```

This is rendered like this:

enumerate(*sequence*[, *start=0*])
Return an iterator that yields tuples of an index and an item of the *sequence*. (And so on.)

The argument of the directive is the *signature* of the object you describe, the content is the documentation for it. Multiple signatures can be given, each in its own line.

The Python domain also happens to be the default domain, so you don't need to prefix the markup with the domain name:

```
.. function:: enumerate(sequence[, start=0])

   ...
```

does the same job if you keep the default setting for the default domain.

There are several more directives for documenting other types of Python objects, for example *py:class* or *py:method*. There is also a cross-referencing *role* for each of these object types. This markup will create a link to the documentation of `enumerate()`:

```
The :py:func:`enumerate` function can be used for ...
```

And here is the proof: A link to *enumerate()*.

Again, the `py:` can be left out if the Python domain is the default one. It doesn't matter which file contains the actual documentation for `enumerate()`; Sphinx will find it and create a link to it.

Each domain will have special rules for how the signatures can look like, and make the formatted output look pretty, or add specific features like links to parameter types, e.g. in the C/C++ domains.

See *Sphinx Domains* for all the available domains and their directives/roles.

2.7 Basic configuration

Earlier we mentioned that the `conf.py` file controls how Sphinx processes your documents. In that file, which is executed as a Python source file, you assign configuration values. For advanced users: since it is executed by Sphinx, you can do non-trivial tasks in it, like extending `sys.path` or importing a module to find out the version you are documenting.

The config values that you probably want to change are already put into the `conf.py` by **`sphinx-quickstart`** and initially commented out (with standard Python syntax: a `#` comments the rest of the line). To change the default value, remove the hash sign and modify the value. To customize a config value that is not automatically added by **`sphinx-quickstart`**, just add an additional assignment.

Keep in mind that the file uses Python syntax for strings, numbers, lists and so on. The file is saved in UTF-8 by default, as indicated by the encoding declaration in the first line. If you use non-ASCII characters in any string value, you need to use Python Unicode strings (like `project = u'Exposé'`).

See *The build configuration file* for documentation of all available config values.

2.8 Autodoc

When documenting Python code, it is common to put a lot of documentation in the source files, in documentation strings. Sphinx supports the inclusion of docstrings from your modules with an *extension* (an extension is a Python module that provides additional features for Sphinx projects) called "autodoc".

In order to use autodoc, you need to activate it in `conf.py` by putting the string `'sphinx.ext.autodoc'` into the list assigned to the *extensions* config value. Then, you have a few additional directives at your disposal.

For example, to document the function `io.open()`, reading its signature and docstring from the source file, you'd write this:

```
.. autofunction:: io.open
```

You can also document whole classes or even modules automatically, using member options for the auto directives, like

```
.. automodule:: io
   :members:
```

autodoc needs to import your modules in order to extract the docstrings. Therefore, you must add the appropriate path to `sys.path` in your `conf.py`.

Warning: *autodoc* **imports** the modules to be documented. If any modules have side effects on import, these will be executed by `autodoc` when `sphinx-build` is run.
If you document scripts (as opposed to library modules), make sure their main routine is protected by a `if __name__ == '__main__'` condition.

See *sphinx.ext.autodoc* for the complete description of the features of autodoc.

2.9 Intersphinx

Many Sphinx documents including the Python documentation[3] are published on the internet. When you want to make links to such documents from your documentation, you can do it with *sphinx.ext.intersphinx*.

In order to use intersphinx, you need to activate it in `conf.py` by putting the string `'sphinx.ext.intersphinx'` into the *extensions* list and set up the *intersphinx_mapping* config value.

For example, to link to `io.open()` in the Python library manual, you need to setup your *intersphinx_mapping* like:

```
intersphinx_mapping = {'python': ('https://docs.python.org/3', None)}
```

And now, you can write a cross-reference like `:py:func:`io.open``. Any cross-reference that has no matching target in the current documentation set, will be looked up in the documentation sets configured in *intersphinx_mapping* (this needs access to the URL in order to download the list of valid targets). Intersphinx also works for some other *domains'* roles including `:ref:`, however it doesn't work for `:doc:` as that is non-domain role.

See *sphinx.ext.intersphinx* for the complete description of the features of intersphinx.

2.10 More topics to be covered

- Other extensions (math, viewcode, doctest)
- Static files
- Selecting a theme
- Templating
- Using extensions
- Writing extensions

[3] https://docs.python.org/3

CHAPTER 3

Invocation of sphinx-quickstart

The **sphinx-quickstart** script generates a Sphinx documentation set. It is called like this:

```
$ sphinx-quickstart [options] [projectdir]
```

where *projectdir* is the Sphinx documentation set directory in which you want to place. If you omit *projectdir*, files are generated into current directory by default.

The **sphinx-quickstart** script has several options:

-q, --quiet
Quiet mode that will skips interactive wizard to specify options. This option requires *-p*, *-a* and *-v* options.

-h, --help, --version
Display usage summary or Sphinx version.

3.1 Structure options

--sep
If specified, separate source and build directories.

--dot=DOT
Inside the root directory, two more directories will be created; "_templates" for custom HTML templates and "_static" for custom stylesheets and other static files. You can enter another prefix (such as ".") to replace the underscore.

3.2 Project basic options

-p PROJECT, **--project**=PROJECT
Project name will be set. (see *project*).

-a AUTHOR, **--author**=AUTHOR
Author names. (see *copyright*).

-v VERSION
Version of project. (see *version*).

-r RELEASE, **--release**=RELEASE
Release of project. (see *release*).

-l LANGUAGE, **--language**=LANGUAGE
Document language. (see *language*).

--suffix=SUFFIX
Source file suffix. (see *source_suffix*).

--master=MASTER
Master document name. (see *master_doc*).

--epub
Use epub.

3.3 Extension options

--ext-autodoc
Enable *sphinx.ext.autodoc* extension.

--ext-doctest
Enable *sphinx.ext.doctest* extension.

--ext-intersphinx
Enable *sphinx.ext.intersphinx* extension.

--ext-todo
Enable *sphinx.ext.todo* extension.

--ext-coverage
Enable *sphinx.ext.coverage* extension.

--ext-pngmath
Enable *sphinx.ext.pngmath* extension.

--ext-mathjax
Enable *sphinx.ext.mathjax* extension.

--ext-ifconfig
Enable *sphinx.ext.ifconfig* extension.

--ext-viewcode
Enable *sphinx.ext.viewcode* extension.

3.4 Makefile and Batchfile creation options

--use-make-mode, --no-use-make-mode
Makefile/make.bat uses (or not use) make-mode. Default is not use.

--makefile, --no-makefile
Create (or not create) makefile.

--batchfile, --no-batchfile
Create (or not create) batchfile

New in version 1.3: Add various options for sphinx-quickstart invocation.

CHAPTER 4

Invocation of sphinx-build

The **sphinx-build** script builds a Sphinx documentation set. It is called like this:

```
$ sphinx-build [options] sourcedir builddir [filenames]
```

where *sourcedir* is the *source directory*, and *builddir* is the directory in which you want to place the built documentation. Most of the time, you don't need to specify any *filenames*.

The **sphinx-build** script has several options:

-b buildername
: The most important option: it selects a builder. The most common builders are:

  html Build HTML pages. This is the default builder.

  dirhtml Build HTML pages, but with a single directory per document. Makes for prettier URLs (no `.html`) if served from a webserver.

  singlehtml Build a single HTML with the whole content.

  htmlhelp, qthelp, devhelp, epub Build HTML files with additional information for building a documentation collection in one of these formats.

  applehelp Build an Apple Help Book. Requires **hiutil** and **codesign**, which are not Open Source and presently only available on Mac OS X 10.6 and higher.

  latex Build LaTeX sources that can be compiled to a PDF document using **pdflatex**.

  man Build manual pages in groff format for UNIX systems.

  texinfo Build Texinfo files that can be processed into Info files using **makeinfo**.

  text Build plain text files.

  gettext Build gettext-style message catalogs (`.pot` files).

  doctest Run all doctests in the documentation, if the *doctest* extension is enabled.

  linkcheck Check the integrity of all external links.

  xml Build Docutils-native XML files.

  pseudoxml Build compact pretty-printed "pseudo-XML" files displaying the internal structure of the intermediate document trees.

  See *Available builders* for a list of all builders shipped with Sphinx. Extensions can add their own builders.

-a
If given, always write all output files. The default is to only write output files for new and changed source files. (This may not apply to all builders.)

-E
Don't use a saved *environment* (the structure caching all cross-references), but rebuild it completely. The default is to only read and parse source files that are new or have changed since the last run.

-t `tag`
Define the tag *tag*. This is relevant for `only` directives that only include their content if this tag is set.

New in version 0.6.

-d `path`
Since Sphinx has to read and parse all source files before it can write an output file, the parsed source files are cached as "doctree pickles". Normally, these files are put in a directory called `.doctrees` under the build directory; with this option you can select a different cache directory (the doctrees can be shared between all builders).

-j `N`
Distribute the build over *N* processes in parallel, to make building on multiprocessor machines more effective. Note that not all parts and not all builders of Sphinx can be parallelized.

New in version 1.2: This option should be considered *experimental*.

-c `path`
Don't look for the `conf.py` in the source directory, but use the given configuration directory instead. Note that various other files and paths given by configuration values are expected to be relative to the configuration directory, so they will have to be present at this location too.

New in version 0.3.

-C
Don't look for a configuration file; only take options via the `-D` option.

New in version 0.5.

-D `setting=value`
Override a configuration value set in the `conf.py` file. The value must be a number, string, list or dictionary value.

For lists, you can separate elements with a comma like this: `-D html_theme_path=path1,path2`.

For dictionary values, supply the setting name and key like this: `-D latex_elements.docclass=scrartcl`.

For boolean values, use `0` or `1` as the value.

Changed in version 0.6: The value can now be a dictionary value.

Changed in version 1.3: The value can now also be a list value.

-A `name=value`
Make the *name* assigned to *value* in the HTML templates.

New in version 0.5.

-n
Run in nit-picky mode. Currently, this generates warnings for all missing references. See the config value `nitpick_ignore` for a way to exclude some references as "known missing".

-N
Do not emit colored output.

-v
Increase verbosity (loglevel). This option can be given up to three times to get more debug logging output. It implies *-T*.

New in version 1.2.

-q
Do not output anything on standard output, only write warnings and errors to standard error.

-Q
Do not output anything on standard output, also suppress warnings. Only errors are written to standard error.

-w `file`
Write warnings (and errors) to the given file, in addition to standard error.

-W
Turn warnings into errors. This means that the build stops at the first warning and `sphinx-build` exits with exit status 1.

-T
Display the full traceback when an unhandled exception occurs. Otherwise, only a summary is displayed and the traceback information is saved to a file for further analysis.

New in version 1.2.

-P
(Useful for debugging only.) Run the Python debugger, `pdb`, if an unhandled exception occurs while building.

-h, --help, --version
Display usage summary or Sphinx version.

New in version 1.2.

You can also give one or more filenames on the command line after the source and build directories. Sphinx will then try to build only these output files (and their dependencies).

4.1 Makefile options

The `Makefile` and `make.bat` files created by **sphinx-quickstart** usually run **sphinx-build** only with the *-b* and *-d* options. However, they support the following variables to customize behavior:

PAPER
The value for *latex_paper_size*.

SPHINXBUILD
The command to use instead of `sphinx-build`.

BUILDDIR
The build directory to use instead of the one chosen in **sphinx-quickstart**.

SPHINXOPTS
Additional options for **sphinx-build**.

CHAPTER 5

Invocation of sphinx-apidoc

The **sphinx-apidoc** generates completely automatic API documentation for a Python package. It is called like this:

```
$ sphinx-apidoc [options] -o outputdir packagedir [pathnames]
```

where *packagedir* is the path to the package to document, and *outputdir* is the directory where the generated sources are placed. Any *pathnames* given are paths to be excluded ignored during generation.

Warning: `sphinx-apidoc` generates reST files that use *`sphinx.ext.autodoc`* to document all found modules. If any modules have side effects on import, these will be executed by `autodoc` when `sphinx-build` is run.
If you document scripts (as opposed to library modules), make sure their main routine is protected by a `if __name__ == '__main__'` condition.

The **sphinx-apidoc** script has several options:

-o `outputdir`
: Gives the directory in which to place the generated output.

-f, --force
: Normally, sphinx-apidoc does not overwrite any files. Use this option to force the overwrite of all files that it generates.

-n, --dry-run
: With this option given, no files will be written at all.

-s `suffix`
: This option selects the file name suffix of output files. By default, this is `rst`.

-d `maxdepth`
: This sets the maximum depth of the table of contents, if one is generated.

-l, --follow-links
: This option makes sphinx-apidoc follow symbolic links when recursing the filesystem to discover packages and modules. You may need it if you want to generate documentation from a source directory managed by collective.recipe.omelette[1]. By default, symbolic links are skipped.

  New in version 1.2.

-T, --no-toc
: This prevents the generation of a table-of-contents file `modules.rst`. This has no effect when *`--full`* is given.

[1] https://pypi.python.org/pypi/collective.recipe.omelette/

-F, --full
This option makes sphinx-apidoc create a full Sphinx project, using the same mechanism as **sphinx-quickstart**. Most configuration values are set to default values, but you can influence the most important ones using the following options.

-M
This option makes sphinx-apidoc put module documentation before submodule documentation.

-H `project`
Sets the project name to put in generated files (see *project*).

-A `author`
Sets the author name(s) to put in generated files (see *copyright*).

-V `version`
Sets the project version to put in generated files (see *version*).

-R `release`
Sets the project release to put in generated files (see *release*).

CHAPTER 6

reStructuredText Primer

This section is a brief introduction to reStructuredText (reST) concepts and syntax, intended to provide authors with enough information to author documents productively. Since reST was designed to be a simple, unobtrusive markup language, this will not take too long.

See also:

The authoritative reStructuredText User Documentation[1]. The "ref" links in this document link to the description of the individual constructs in the reST reference.

6.1 Paragraphs

The paragraph (ref[2]) is the most basic block in a reST document. Paragraphs are simply chunks of text separated by one or more blank lines. As in Python, indentation is significant in reST, so all lines of the same paragraph must be left-aligned to the same level of indentation.

6.2 Inline markup

The standard reST inline markup is quite simple: use

- one asterisk: `*text*` for emphasis (italics),
- two asterisks: `**text**` for strong emphasis (boldface), and
- backquotes: ``` ``text`` ``` for code samples.

If asterisks or backquotes appear in running text and could be confused with inline markup delimiters, they have to be escaped with a backslash.

Be aware of some restrictions of this markup:

- it may not be nested,
- content may not start or end with whitespace: `* text*` is wrong,
- it must be separated from surrounding text by non-word characters. Use a backslash escaped space to work around that: `thisis\ *one*\ word`.

[1] http://docutils.sourceforge.net/rst.html
[2] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#paragraphs

These restrictions may be lifted in future versions of the docutils.

reST also allows for custom "interpreted text roles", which signify that the enclosed text should be interpreted in a specific way. Sphinx uses this to provide semantic markup and cross-referencing of identifiers, as described in the appropriate section. The general syntax is `:rolename:`content``.

Standard reST provides the following roles:

- emphasis[3] – alternate spelling for `*emphasis*`
- strong[4] – alternate spelling for `**strong**`
- literal[5] – alternate spelling for ``` ``literal`` ```
- subscript[6] – subscript text
- superscript[7] – superscript text
- title-reference[8] – for titles of books, periodicals, and other materials

See *Inline markup* for roles added by Sphinx.

6.3 Lists and Quote-like blocks

List markup (ref[9]) is natural: just place an asterisk at the start of a paragraph and indent properly. The same goes for numbered lists; they can also be autonumbered using a `#` sign:

```
* This is a bulleted list.
* It has two items, the second
  item uses two lines.

1. This is a numbered list.
2. It has two items too.

#. This is a numbered list.
#. It has two items too.
```

Nested lists are possible, but be aware that they must be separated from the parent list items by blank lines:

```
* this is
* a list

  * with a nested list
  * and some subitems

* and here the parent list continues
```

Definition lists (ref[10]) are created as follows:

```
term (up to a line of text)
   Definition of the term, which must be indented

```

[3] http://docutils.sourceforge.net/docs/ref/rst/roles.html#emphasis
[4] http://docutils.sourceforge.net/docs/ref/rst/roles.html#strong
[5] http://docutils.sourceforge.net/docs/ref/rst/roles.html#literal
[6] http://docutils.sourceforge.net/docs/ref/rst/roles.html#subscript
[7] http://docutils.sourceforge.net/docs/ref/rst/roles.html#superscript
[8] http://docutils.sourceforge.net/docs/ref/rst/roles.html#title-reference
[9] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#bullet-lists
[10] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#definition-lists

```
   and can even consist of multiple paragraphs

next term
   Description.
```

Note that the term cannot have more than one line of text.

Quoted paragraphs (ref[11]) are created by just indenting them more than the surrounding paragraphs.

Line blocks (ref[12]) are a way of preserving line breaks:

```
| These lines are
| broken exactly like in
| the source file.
```

There are also several more special blocks available:

- field lists (ref[13])
- option lists (ref[14])
- quoted literal blocks (ref[15])
- doctest blocks (ref[16])

6.4 Source Code

Literal code blocks (ref[17]) are introduced by ending a paragraph with the special marker `::`. The literal block must be indented (and, like all paragraphs, separated from the surrounding ones by blank lines):

```
This is a normal text paragraph. The next paragraph is a code sample::

   It is not processed in any way, except
   that the indentation is removed.

   It can span multiple lines.

This is a normal text paragraph again.
```

The handling of the `::` marker is smart:

- If it occurs as a paragraph of its own, that paragraph is completely left out of the document.
- If it is preceded by whitespace, the marker is removed.
- If it is preceded by non-whitespace, the marker is replaced by a single colon.

That way, the second sentence in the above example's first paragraph would be rendered as "The next paragraph is a code sample:".

[11] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#block-quotes
[12] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#line-blocks
[13] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#field-lists
[14] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#option-lists
[15] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#quoted-literal-blocks
[16] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#doctest-blocks
[17] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#literal-blocks

6.5 Tables

Two forms of tables are supported. For *grid tables* (ref[18]), you have to "paint" the cell grid yourself. They look like this:

```
+------------------------+------------+----------+----------+
| Header row, column 1   | Header 2   | Header 3 | Header 4 |
| (header rows optional) |            |          |          |
+========================+============+==========+==========+
| body row 1, column 1   | column 2   | column 3 | column 4 |
+------------------------+------------+----------+----------+
| body row 2             | ...        | ...      |          |
+------------------------+------------+----------+----------+
```

Simple tables (ref[19]) are easier to write, but limited: they must contain more than one row, and the first column cannot contain multiple lines. They look like this:

```
=====  =====  =======
A      B      A and B
=====  =====  =======
False  False  False
True   False  False
False  True   False
True   True   True
=====  =====  =======
```

6.6 Hyperlinks

6.6.1 External links

Use `'Link text <http://example.com/>'_` for inline web links. If the link text should be the web address, you don't need special markup at all, the parser finds links and mail addresses in ordinary text.

You can also separate the link and the target definition (ref[20]), like this:

```
This is a paragraph that contains `a link`_.

.. _a link: http://example.com/
```

6.6.2 Internal links

Internal linking is done via a special reST role provided by Sphinx, see the section on specific markup, *Cross-referencing arbitrary locations*.

6.7 Sections

Section headers (ref[21]) are created by underlining (and optionally overlining) the section title with a punctuation character, at least as long as the text:

[18] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#grid-tables
[19] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#simple-tables
[20] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#hyperlink-targets
[21] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#sections

```
=================
This is a heading
=================
```

Normally, there are no heading levels assigned to certain characters as the structure is determined from the succession of headings. However, for the Python documentation, this convention is used which you may follow:

- # with overline, for parts
- * with overline, for chapters
- =, for sections
- -, for subsections
- ^, for subsubsections
- ", for paragraphs

Of course, you are free to use your own marker characters (see the reST documentation), and use a deeper nesting level, but keep in mind that most target formats (HTML, LaTeX) have a limited supported nesting depth.

6.8 Explicit Markup

"Explicit markup" (ref[22]) is used in reST for most constructs that need special handling, such as footnotes, specially-highlighted paragraphs, comments, and generic directives.

An explicit markup block begins with a line starting with `..` followed by whitespace and is terminated by the next paragraph at the same level of indentation. (There needs to be a blank line between explicit markup and normal paragraphs. This may all sound a bit complicated, but it is intuitive enough when you write it.)

6.9 Directives

A directive (ref[23]) is a generic block of explicit markup. Besides roles, it is one of the extension mechanisms of reST, and Sphinx makes heavy use of it.

Docutils supports the following directives:

- Admonitions: attention[24], caution[25], danger[26], error[27], hint[28], important[29], note[30], tip[31], warning[32] and the generic admonition[33]. (Most themes style only "note" and "warning" specially.)
- Images:

[22] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#explicit-markup-blocks
[23] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#directives
[24] http://docutils.sourceforge.net/docs/ref/rst/directives.html#attention
[25] http://docutils.sourceforge.net/docs/ref/rst/directives.html#caution
[26] http://docutils.sourceforge.net/docs/ref/rst/directives.html#danger
[27] http://docutils.sourceforge.net/docs/ref/rst/directives.html#error
[28] http://docutils.sourceforge.net/docs/ref/rst/directives.html#hint
[29] http://docutils.sourceforge.net/docs/ref/rst/directives.html#important
[30] http://docutils.sourceforge.net/docs/ref/rst/directives.html#note
[31] http://docutils.sourceforge.net/docs/ref/rst/directives.html#tip
[32] http://docutils.sourceforge.net/docs/ref/rst/directives.html#warning
[33] http://docutils.sourceforge.net/docs/ref/rst/directives.html#admonitions

  - image[34] (see also *Images* below)
  - figure[35] (an image with caption and optional legend)
- Additional body elements:
  - contents[36] (a local, i.e. for the current file only, table of contents)
  - container[37] (a container with a custom class, useful to generate an outer `<div>` in HTML)
  - rubric[38] (a heading without relation to the document sectioning)
  - topic[39], sidebar[40] (special highlighted body elements)
  - parsed-literal[41] (literal block that supports inline markup)
  - epigraph[42] (a block quote with optional attribution line)
  - highlights[43], pull-quote[44] (block quotes with their own class attribute)
  - compound[45] (a compound paragraph)
- Special tables:
  - table[46] (a table with title)
  - csv-table[47] (a table generated from comma-separated values)
  - list-table[48] (a table generated from a list of lists)
- Special directives:
  - raw[49] (include raw target-format markup)
  - include[50] (include reStructuredText from another file) – in Sphinx, when given an absolute include file path, this directive takes it as relative to the source directory
  - class[51] (assign a class attribute to the next element) [52]
- HTML specifics:
  - meta[53] (generation of HTML `<meta>` tags)
  - title[54] (override document title)
- Influencing markup:

[34] http://docutils.sourceforge.net/docs/ref/rst/directives.html#image
[35] http://docutils.sourceforge.net/docs/ref/rst/directives.html#figure
[36] http://docutils.sourceforge.net/docs/ref/rst/directives.html#table-of-contents
[37] http://docutils.sourceforge.net/docs/ref/rst/directives.html#container
[38] http://docutils.sourceforge.net/docs/ref/rst/directives.html#rubric
[39] http://docutils.sourceforge.net/docs/ref/rst/directives.html#topic
[40] http://docutils.sourceforge.net/docs/ref/rst/directives.html#sidebar
[41] http://docutils.sourceforge.net/docs/ref/rst/directives.html#parsed-literal
[42] http://docutils.sourceforge.net/docs/ref/rst/directives.html#epigraph
[43] http://docutils.sourceforge.net/docs/ref/rst/directives.html#highlights
[44] http://docutils.sourceforge.net/docs/ref/rst/directives.html#pull-quote
[45] http://docutils.sourceforge.net/docs/ref/rst/directives.html#compound-paragraph
[46] http://docutils.sourceforge.net/docs/ref/rst/directives.html#table
[47] http://docutils.sourceforge.net/docs/ref/rst/directives.html#csv-table
[48] http://docutils.sourceforge.net/docs/ref/rst/directives.html#list-table
[49] http://docutils.sourceforge.net/docs/ref/rst/directives.html#raw-data-pass-through
[50] http://docutils.sourceforge.net/docs/ref/rst/directives.html#include
[51] http://docutils.sourceforge.net/docs/ref/rst/directives.html#class
[52] When the default domain contains a `class` directive, this directive will be shadowed. Therefore, Sphinx re-exports it as `rst-class`.
[53] http://docutils.sourceforge.net/docs/ref/rst/directives.html#meta
[54] http://docutils.sourceforge.net/docs/ref/rst/directives.html#metadata-document-title

- default-role[55] (set a new default role)
- role[56] (create a new role)

Since these are only per-file, better use Sphinx's facilities for setting the `default_role`.

Do *not* use the directives sectnum[57], header[58] and footer[59].

Directives added by Sphinx are described in *Sphinx Markup Constructs*.

Basically, a directive consists of a name, arguments, options and content. (Keep this terminology in mind, it is used in the next chapter describing custom directives.) Looking at this example,

```
.. function:: foo(x)
              foo(y, z)
   :module: some.module.name

   Return a line of text input from the user.
```

`function` is the directive name. It is given two arguments here, the remainder of the first line and the second line, as well as one option `module` (as you can see, options are given in the lines immediately following the arguments and indicated by the colons). Options must be indented to the same level as the directive content.

The directive content follows after a blank line and is indented relative to the directive start.

6.10 Images

reST supports an image directive (ref[60]), used like so:

```
.. image:: gnu.png
   (options)
```

When used within Sphinx, the file name given (here `gnu.png`) must either be relative to the source file, or absolute which means that they are relative to the top source directory. For example, the file `sketch/spam.rst` could refer to the image `images/spam.png` as `../images/spam.png` or `/images/spam.png`.

Sphinx will automatically copy image files over to a subdirectory of the output directory on building (e.g. the `_static` directory for HTML output.)

Interpretation of image size options (`width` and `height`) is as follows: if the size has no unit or the unit is pixels, the given size will only be respected for output channels that support pixels (i.e. not in LaTeX output). Other units (like `pt` for points) will be used for HTML and LaTeX output.

Sphinx extends the standard docutils behavior by allowing an asterisk for the extension:

```
.. image:: gnu.*
```

Sphinx then searches for all images matching the provided pattern and determines their type. Each builder then chooses the best image out of these candidates. For instance, if the file name `gnu.*` was given and two files `gnu.pdf` and `gnu.png` existed in the source tree, the LaTeX builder would choose the former,

[55] http://docutils.sourceforge.net/docs/ref/rst/directives.html#default-role
[56] http://docutils.sourceforge.net/docs/ref/rst/directives.html#role
[57] http://docutils.sourceforge.net/docs/ref/rst/directives.html#sectnum
[58] http://docutils.sourceforge.net/docs/ref/rst/directives.html#header
[59] http://docutils.sourceforge.net/docs/ref/rst/directives.html#footer
[60] http://docutils.sourceforge.net/docs/ref/rst/directives.html#image

while the HTML builder would prefer the latter. Supported image types and choosing priority are defined at *Available builders*.

Note that image file names should not contain spaces.

Changed in version 0.4: Added the support for file names ending in an asterisk.

Changed in version 0.6: Image paths can now be absolute.

6.11 Footnotes

For footnotes (ref[61]), use `[#name]_` to mark the footnote location, and add the footnote body at the bottom of the document after a "Footnotes" rubric heading, like so:

```
Lorem ipsum [#f1]_ dolor sit amet ... [#f2]_

.. rubric:: Footnotes

.. [#f1] Text of the first footnote.
.. [#f2] Text of the second footnote.
```

You can also explicitly number the footnotes (`[1]_`) or use auto-numbered footnotes without names (`[#]_`).

6.12 Citations

Standard reST citations (ref[62]) are supported, with the additional feature that they are "global", i.e. all citations can be referenced from all files. Use them like so:

```
Lorem ipsum [Ref]_ dolor sit amet.

.. [Ref] Book or article reference, URL or whatever.
```

Citation usage is similar to footnote usage, but with a label that is not numeric or begins with `#`.

6.13 Substitutions

reST supports "substitutions" (ref[63]), which are pieces of text and/or markup referred to in the text by `|name|`. They are defined like footnotes with explicit markup blocks, like this:

```
.. |name| replace:: replacement *text*
```

or this:

```
.. |caution| image:: warning.png
              :alt: Warning!
```

See the reST reference for substitutions[64] for details.

If you want to use some substitutions for all documents, put them into *`rst_prolog`* or put them into a separate file and include it into all documents you want to use them in, using the `include` directive. (Be

[61] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#footnotes
[62] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#citations
[63] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#substitution-definitions
[64] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#substitution-definitions

sure to give the include file a file name extension differing from that of other source files, to avoid Sphinx finding it as a standalone document.)

Sphinx defines some default substitutions, see *Substitutions*.

6.14 Comments

Every explicit markup block which isn't a valid markup construct (like the footnotes above) is regarded as a comment (ref[65]). For example:

```
.. This is a comment.
```

You can indent text after a comment start to form multiline comments:

```
..
   This whole indented block
   is a comment.

   Still in the comment.
```

6.15 Source encoding

Since the easiest way to include special characters like em dashes or copyright signs in reST is to directly write them as Unicode characters, one has to specify an encoding. Sphinx assumes source files to be encoded in UTF-8 by default; you can change this with the `source_encoding` config value.

6.16 Gotchas

There are some problems one commonly runs into while authoring reST documents:

- **Separation of inline markup:** As said above, inline markup spans must be separated from the surrounding text by non-word characters, you have to use a backslash-escaped space to get around that. See the reference[66] for the details.
- **No nested inline markup:** Something like `*see :func:`foo`*` is not possible.

[65] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#comments
[66] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html#substitution-definitions

CHAPTER 7

Sphinx Markup Constructs

Sphinx adds a lot of new directives and interpreted text roles to standard reST markup[1]. This section contains the reference material for these facilities.

7.1 The TOC tree

Since reST does not have facilities to interconnect several documents, or split documents into multiple output files, Sphinx uses a custom directive to add relations between the single files the documentation is made of, as well as tables of contents. The `toctree` directive is the central element.

Note: Simple "inclusion" of one file in another can be done with the include[2] directive.

`.. toctree::`

This directive inserts a "TOC tree" at the current location, using the individual TOCs (including "sub-TOC trees") of the documents given in the directive body. Relative document names (not beginning with a slash) are relative to the document the directive occurs in, absolute names are relative to the source directory. A numeric `maxdepth` option may be given to indicate the depth of the tree; by default, all levels are included. [3]

Consider this example (taken from the Python docs' library reference index):

```
.. toctree::
   :maxdepth: 2

   intro
   strings
   datatypes
   numeric
   (many more documents listed here)
```

This accomplishes two things:

- Tables of contents from all those documents are inserted, with a maximum depth of two, that means one nested heading. `toctree` directives in those documents are also taken into account.

[1] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html
[2] http://docutils.sourceforge.net/docs/ref/rst/directives.html#include
[3] The `maxdepth` option does not apply to the LaTeX writer, where the whole table of contents will always be presented at the begin of the document, and its depth is controlled by the `tocdepth` counter, which you can reset in your *`latex_preamble`* config value using e.g. `\setcounter{tocdepth}{2}`.

- Sphinx knows that the relative order of the documents `intro`, `strings` and so forth, and it knows that they are children of the shown document, the library index. From this information it generates "next chapter", "previous chapter" and "parent chapter" links.

Entries

Document titles in the *`toctree`* will be automatically read from the title of the referenced document. If that isn't what you want, you can specify an explicit title and target using a similar syntax to reST hyperlinks (and Sphinx's *cross-referencing syntax*). This looks like:

```
.. toctree::

   intro
   All about strings <strings>
   datatypes
```

The second line above will link to the `strings` document, but will use the title "All about strings" instead of the title of the `strings` document.

You can also add external links, by giving an HTTP URL instead of a document name.

Section numbering

If you want to have section numbers even in HTML output, give the **toplevel** toctree a `numbered` option. For example:

```
.. toctree::
   :numbered:

   foo
   bar
```

Numbering then starts at the heading of `foo`. Sub-toctrees are automatically numbered (don't give the `numbered` flag to those).

Numbering up to a specific depth is also possible, by giving the depth as a numeric argument to `numbered`.

Additional options

You can use `caption` option to provide a toctree caption and you can use `name` option to provide implicit target name that can be referenced by using *`ref`*:

```
.. toctree::
   :caption: Table of Contents
   :name: mastertoc

   foo
```

If you want only the titles of documents in the tree to show up, not other headings of the same level, you can use the `titlesonly` option:

```
.. toctree::
   :titlesonly:

   foo
   bar
```

You can use "globbing" in toctree directives, by giving the `glob` flag option. All entries are then matched against the list of available documents, and matches are inserted into the list alphabetically. Example:

```
.. toctree::
   :glob:

   intro*
   recipe/*
   *
```

This includes first all documents whose names start with `intro`, then all documents in the `recipe` folder, then all remaining documents (except the one containing the directive, of course.) [4]

The special entry name `self` stands for the document containing the toctree directive. This is useful if you want to generate a "sitemap" from the toctree.

You can also give a "hidden" option to the directive, like this:

```
.. toctree::
   :hidden:

   doc_1
   doc_2
```

This will still notify Sphinx of the document hierarchy, but not insert links into the document at the location of the directive – this makes sense if you intend to insert these links yourself, in a different style, or in the HTML sidebar.

In cases where you want to have only one top-level toctree and hide all other lower level toctrees you can add the "includehidden" option to the top-level toctree entry:

```
.. toctree::
   :includehidden:

   doc_1
   doc_2
```

All other toctree entries can then be eliminated by the "hidden" option.

In the end, all documents in the *source directory* (or subdirectories) must occur in some `toctree` directive; Sphinx will emit a warning if it finds a file that is not included, because that means that this file will not be reachable through standard navigation.

Use *`exclude_patterns`* to explicitly exclude documents or directories from building completely. Use *the "orphan" metadata* to let a document be built, but notify Sphinx that it is not reachable via a toctree.

The "master document" (selected by *`master_doc`*) is the "root" of the TOC tree hierarchy. It can be used as the documentation's main page, or as a "full table of contents" if you don't give a `maxdepth` option.

Changed in version 0.3: Added "globbing" option.

Changed in version 0.6: Added "numbered" and "hidden" options as well as external links and support for "self" references.

Changed in version 1.0: Added "titlesonly" option.

Changed in version 1.1: Added numeric argument to "numbered".

Changed in version 1.2: Added "includehidden" option.

Changed in version 1.3: Added "caption" and "name" option.

[4] A note on available globbing syntax: you can use the standard shell constructs `*`, `?`, `[...]` and `[!...]` with the feature that these all don't match slashes. A double star `**` can be used to match any sequence of characters *including* slashes.

7.1.1 Special names

Sphinx reserves some document names for its own use; you should not try to create documents with these names – it will cause problems.

The special document names (and pages generated for them) are:

- `genindex`, `modindex`, `search`

  These are used for the general index, the Python module index, and the search page, respectively.

  The general index is populated with entries from modules, all index-generating *object descriptions*, and from `index` directives.

  The Python module index contains one entry per `py:module` directive.

  The search page contains a form that uses the generated JSON search index and JavaScript to full-text search the generated documents for search words; it should work on every major browser that supports modern JavaScript.

- every name beginning with _

  Though only few such names are currently used by Sphinx, you should not create documents or document-containing directories with such names. (Using _ as a prefix for a custom template directory is fine.)

Warning: Be careful with unusual characters in filenames. Some formats may interpret these characters in unexpected ways:

- Do not use the colon `:` for HTML based formats. Links to other parts may not work.
- Do not use the plus `+` for the ePub format. Some resources may not be found.

7.2 Paragraph-level markup

These directives create short paragraphs and can be used inside information units as well as normal text:

`.. note::`
An especially important bit of information about an API that a user should be aware of when using whatever bit of API the note pertains to. The content of the directive should be written in complete sentences and include all appropriate punctuation.

Example:

```
.. note::

   This function is not suitable for sending spam e-mails.
```

`.. warning::`
An important bit of information about an API that a user should be very aware of when using whatever bit of API the warning pertains to. The content of the directive should be written in complete sentences and include all appropriate punctuation. This differs from `note` in that it is recommended over `note` for information regarding security.

`.. versionadded:: version`
This directive documents the version of the project which added the described feature to the library or C API. When this applies to an entire module, it should be placed at the top of the module section before any prose.

The first argument must be given and is the version in question; you can add a second argument consisting of a *brief* explanation of the change.

Example:

```
.. versionadded:: 2.5
   The *spam* parameter.
```

Note that there must be no blank line between the directive head and the explanation; this is to make these blocks visually continuous in the markup.

`.. versionchanged:: version`
Similar to *versionadded*, but describes when and what changed in the named feature in some way (new parameters, changed side effects, etc.).

`.. deprecated:: version`
Similar to *versionchanged*, but describes when the feature was deprecated. An explanation can also be given, for example to inform the reader what should be used instead. Example:

```
.. deprecated:: 3.1
   Use :func:`spam` instead.
```

`.. seealso::`
Many sections include a list of references to module documentation or external documents. These lists are created using the *seealso* directive.

The *seealso* directive is typically placed in a section just before any subsections. For the HTML output, it is shown boxed off from the main flow of the text.

The content of the *seealso* directive should be a reST definition list. Example:

```
.. seealso::

   Module :py:mod:`zipfile`
      Documentation of the :py:mod:`zipfile` standard module.

   `GNU tar manual, Basic Tar Format <http://link>`_
      Documentation for tar archive files, including GNU tar extensions.
```

There's also a "short form" allowed that looks like this:

```
.. seealso:: modules :py:mod:`zipfile`, :py:mod:`tarfile`
```

New in version 0.5: The short form.

`.. rubric:: title`
This directive creates a paragraph heading that is not used to create a table of contents node.

Note: If the *title* of the rubric is "Footnotes" (or the selected language's equivalent), this rubric is ignored by the LaTeX writer, since it is assumed to only contain footnote definitions and therefore would create an empty heading.

`.. centered::`
This directive creates a centered boldfaced line of text. Use it as follows:

```
.. centered:: LICENSE AGREEMENT
```

Deprecated since version 1.1: This presentation-only directive is a legacy from older versions. Use a `rst-class` directive instead and add an appropriate style.

.. hlist::
This directive must contain a bullet list. It will transform it into a more compact list by either distributing more than one item horizontally, or reducing spacing between items, depending on the builder.

For builders that support the horizontal distribution, there is a `columns` option that specifies the number of columns; it defaults to 2. Example:

```
.. hlist::
   :columns: 3

   * A list of
   * short items
   * that should be
   * displayed
   * horizontally
```

New in version 0.6.

7.3 Table-of-contents markup

The *toctree* directive, which generates tables of contents of subdocuments, is described in *The TOC tree*.

For local tables of contents, use the standard reST contents directive[5].

7.4 Glossary

.. glossary::
This directive must contain a reST definition-list-like markup with terms and definitions. The definitions will then be referencable with the *term* role. Example:

```
.. glossary::

   environment
      A structure where information about all documents under the root is
      saved, and used for cross-referencing.  The environment is pickled
      after the parsing stage, so that successive runs only need to read
      and parse new and changed documents.

   source directory
      The directory which, including its subdirectories, contains all
      source files for one Sphinx project.
```

In contrast to regular definition lists, *multiple* terms per entry are allowed, and inline markup is allowed in terms. You can link to all of the terms. For example:

```
.. glossary::

   term 1
   term 2
      Definition of both terms.
```

(When the glossary is sorted, the first term determines the sort order.)

New in version 0.6: You can now give the glossary directive a `:sorted:` flag that will automatically sort the entries alphabetically.

[5] http://docutils.sourceforge.net/docs/ref/rst/directives.html#table-of-contents

Changed in version 1.1: Now supports multiple terms and inline markup in terms.

7.5 Grammar production displays

Special markup is available for displaying the productions of a formal grammar. The markup is simple and does not attempt to model all aspects of BNF (or any derived forms), but provides enough to allow context-free grammars to be displayed in a way that causes uses of a symbol to be rendered as hyperlinks to the definition of the symbol. There is this directive:

`.. productionlist:: [name]`
: This directive is used to enclose a group of productions. Each production is given on a single line and consists of a name, separated by a colon from the following definition. If the definition spans multiple lines, each continuation line must begin with a colon placed at the same column as in the first line.

  The argument to *productionlist* serves to distinguish different sets of production lists that belong to different grammars.

  Blank lines are not allowed within `productionlist` directive arguments.

  The definition can contain token names which are marked as interpreted text (e.g. `sum ::= 'integer' "+" 'integer'`) – this generates cross-references to the productions of these tokens. Outside of the production list, you can reference to token productions using *token*.

  Note that no further reST parsing is done in the production, so that you don't have to escape `*` or `|` characters.

The following is an example taken from the Python Reference Manual:

```
.. productionlist::
   try_stmt: try1_stmt | try2_stmt
   try1_stmt: "try" ":" `suite`
            : ("except" [`expression` ["," `target`]] ":" `suite`)+
            : ["else" ":" `suite`]
            : ["finally" ":" `suite`]
   try2_stmt: "try" ":" `suite`
            : "finally" ":" `suite`
```

7.6 Showing code examples

Examples of Python source code or interactive sessions are represented using standard reST literal blocks. They are started by a `::` at the end of the preceding paragraph and delimited by indentation.

Representing an interactive session requires including the prompts and output along with the Python code. No special markup is required for interactive sessions. After the last line of input or output presented, there should not be an "unused" primary prompt; this is an example of what *not* to do:

```
>>> 1 + 1
2
>>>
```

Syntax highlighting is done with Pygments[6] (if it's installed) and handled in a smart way:

- There is a "highlighting language" for each source file. Per default, this is `'python'` as the majority of files will have to highlight Python snippets, but the doc-wide default can be set with the *highlight_language* config value.

[6] http://pygments.org

- Within Python highlighting mode, interactive sessions are recognized automatically and highlighted appropriately. Normal Python code is only highlighted if it is parseable (so you can use Python as the default, but interspersed snippets of shell commands or other code blocks will not be highlighted as Python).
- The highlighting language can be changed using the `highlight` directive, used as follows:

  `.. highlight:: language`
  Example:

```
.. highlight:: c
```

  This language is used until the next `highlight` directive is encountered.

- For documents that have to show snippets in different languages, there's also a *`code-block`* directive that is given the highlighting language directly:

  `.. code-block:: language`
  Use it like this:

```
.. code-block:: ruby

   Some Ruby code.
```

  The directive's alias name `sourcecode` works as well.

- The valid values for the highlighting language are:
  - `none` (no highlighting)
  - `python` (the default when *`highlight_language`* isn't set)
  - `guess` (let Pygments guess the lexer based on contents, only works with certain well-recognizable languages)
  - `rest`
  - `c`
  - ... and any other lexer alias that Pygments supports[7].
- If highlighting with the selected language fails (i.e. Pygments emits an "Error" token), the block is not highlighted in any way.

7.6.1 Line numbers

If installed, Pygments can generate line numbers for code blocks. For automatically-highlighted blocks (those started by `::`), line numbers must be switched on in a *`highlight`* directive, with the `linenothreshold` option:

```
.. highlight:: python
   :linenothreshold: 5
```

This will produce line numbers for all code blocks longer than five lines.

For *`code-block`* blocks, a `linenos` flag option can be given to switch on line numbers for the individual block:

[7] http://pygments.org/docs/lexers/

```
.. code-block:: ruby
   :linenos:

   Some more Ruby code.
```

The first line number can be selected with the `lineno-start` option. If present, `linenos` is automatically activated as well.

```
10  Some more Ruby code, with line numbering starting at 10.
```

Additionally, an `emphasize-lines` option can be given to have Pygments emphasize particular lines:

```
.. code-block:: python
   :emphasize-lines: 3,5

   def some_function():
       interesting = False
       print 'This line is highlighted.'
       print 'This one is not...'
       print '...but this one is.'
```

Changed in version 1.1: `emphasize-lines` has been added.

Changed in version 1.3: `lineno-start` has been added.

7.6.2 Includes

`.. literalinclude::` filename

Longer displays of verbatim text may be included by storing the example text in an external file containing only plain text. The file may be included using the `literalinclude` directive. [8] For example, to include the Python source file `example.py`, use:

```
.. literalinclude:: example.py
```

The file name is usually relative to the current file's path. However, if it is absolute (starting with `/`), it is relative to the top source directory.

Tabs in the input are expanded if you give a `tab-width` option with the desired tab width.

Like *code-block*, the directive supports the `linenos` flag option to switch on line numbers, the `lineno-start` option to select the first line number, the `emphasize-lines` option to emphasize particular lines, and a `language` option to select a language different from the current file's standard language. Example with options:

```
.. literalinclude:: example.rb
   :language: ruby
   :emphasize-lines: 12,15-18
   :linenos:
```

Include files are assumed to be encoded in the *source_encoding*. If the file has a different encoding, you can specify it with the `encoding` option:

```
.. literalinclude:: example.py
   :encoding: latin-1
```

The directive also supports including only parts of the file. If it is a Python module, you can select a class, function or method to include using the `pyobject` option:

[8] There is a standard `.. include` directive, but it raises errors if the file is not found. This one only emits a warning.

```
.. literalinclude:: example.py
   :pyobject: Timer.start
```

This would only include the code lines belonging to the `start()` method in the `Timer` class within the file.

Alternately, you can specify exactly which lines to include by giving a `lines` option:

```
.. literalinclude:: example.py
   :lines: 1,3,5-10,20-
```

This includes the lines 1, 3, 5 to 10 and lines 20 to the last line.

Another way to control which part of the file is included is to use the `start-after` and `end-before` options (or only one of them). If `start-after` is given as a string option, only lines that follow the first line containing that string are included. If `end-before` is given as a string option, only lines that precede the first lines containing that string are included.

When specifying particular parts of a file to display, it can be useful to display exactly which lines are being presented. This can be done using the `lineno-match` option.

You can prepend and/or append a line to the included code, using the `prepend` and `append` option, respectively. This is useful e.g. for highlighting PHP code that doesn't include the `<?php/?>` markers.

If you want to show the diff of the code, you can specify the old file by giving a `diff` option:

```
.. literalinclude:: example.py
   :diff: example.py.orig
```

This shows the diff between example.py and example.py.orig with unified diff format.

New in version 0.4.3: The `encoding` option.

New in version 0.6: The `pyobject`, `lines`, `start-after` and `end-before` options, as well as support for absolute filenames.

New in version 1.0: The `prepend` and `append` options, as well as `tab-width`.

New in version 1.3: The `diff` option. The `lineno-match` option.

7.6.3 Caption and name

New in version 1.3.

A `caption` option can be given to show that name before the code block. A `name` option can be provided implicit target name that can be referenced by using *`ref`*. For example:

```
.. code-block:: python
   :caption: this.py
   :name: this-py

   print 'Explicit is better than implicit.'
```

`literalinclude` also supports the `caption` and `name` option. `caption` has a additional feature that if you leave the value empty, the shown filename will be exactly the one given as an argument.

7.6.4 Dedent

New in version 1.3.

A `dedent` option can be given to strip a precedence characters from the code block. For example:

```
.. literalinclude:: example.rb
   :language: ruby
   :dedent: 4
   :lines: 10-15
```

code-block also supports the `dedent` option.

7.7 Inline markup

Sphinx uses interpreted text roles to insert semantic markup into documents. They are written as `:rolename:`content``.

Note: The default role (`content`) has no special meaning by default. You are free to use it for anything you like, e.g. variable names; use the *default_role* config value to set it to a known role – the *any* role to find anything or the *py:obj* role to find Python objects are very useful for this.

See *Sphinx Domains* for roles added by domains.

7.7.1 Cross-referencing syntax

Cross-references are generated by many semantic interpreted text roles. Basically, you only need to write `:role:`target``, and a link will be created to the item named *target* of the type indicated by *role*. The link's text will be the same as *target*.

There are some additional facilities, however, that make cross-referencing roles more versatile:

- You may supply an explicit title and reference target, like in reST direct hyperlinks: `:role:`title <target>`` will refer to *target*, but the link text will be *title*.
- If you prefix the content with `!`, no reference/hyperlink will be created.
- If you prefix the content with `~`, the link text will only be the last component of the target. For example, `:py:meth:`~Queue.Queue.get`` will refer to `Queue.Queue.get` but only display `get` as the link text. This does not work with all cross-reference roles, but is domain specific.

  In HTML output, the link's `title` attribute (that is e.g. shown as a tool-tip on mouse-hover) will always be the full target name.

Cross-referencing anything

:any:
New in version 1.3.

This convenience role tries to do its best to find a valid target for its reference text.

- First, it tries standard cross-reference targets that would be referenced by *doc*, *ref* or *option*.

  Custom objects added to the standard domain by extensions (see *add_object_type()*) are also searched.

- Then, it looks for objects (targets) in all loaded domains. It is up to the domains how specific a match must be. For example, in the Python domain a reference of `:any:`Builder`` would match the `sphinx.builders.Builder` class.

If none or multiple targets are found, a warning will be emitted. In the case of multiple targets, you can change "any" to a specific role.

This role is a good candidate for setting *default_role*. If you do, you can write cross-references without a lot of markup overhead. For example, in this Python function documentation

```
.. function:: install()

   This function installs a `handler` for every signal known by the
   `signal` module.  See the section `about-signals` for more information.
```

there could be references to a glossary term (usually `:term:`handler``), a Python module (usually `:py:mod:`signal`` or `:mod:`signal``) and a section (usually `:ref:`about-signals``).

The *any* role also works together with the *intersphinx* extension: when no local cross-reference is found, all object types of intersphinx inventories are also searched.

Cross-referencing objects

These roles are described with their respective domains:

- *Python*
- *C*
- *C++*
- *JavaScript*
- *ReST*

Cross-referencing arbitrary locations

:ref:

To support cross-referencing to arbitrary locations in any document, the standard reST labels are used. For this to work label names must be unique throughout the entire documentation. There are two ways in which you can refer to labels:

- If you place a label directly before a section title, you can reference to it with `:ref:`label-name``. Example:

```
.. _my-reference-label:

Section to cross-reference
--------------------------

This is the text of the section.

It refers to the section itself, see :ref:`my-reference-label`.
```

The `:ref:` role would then generate a link to the section, with the link title being "Section to cross-reference". This works just as well when section and reference are in different source files.

Automatic labels also work with figures: given

```
.. _my-figure:

.. figure:: whatever

   Figure caption
```

a reference `:ref:'my-figure'` would insert a reference to the figure with link text "Figure caption".

The same works for tables that are given an explicit caption using the table[9] directive.

- Labels that aren't placed before a section title can still be referenced to, but you must give the link an explicit title, using this syntax: `:ref:'Link title <label-name>'`.

Using *ref* is advised over standard reStructuredText links to sections (like `'Section title'_`) because it works across files, when section headings are changed, and for all builders that support cross-references.

Cross-referencing documents

New in version 0.6.

There is also a way to directly link to documents:

:doc:
Link to the specified document; the document name can be specified in absolute or relative fashion. For example, if the reference `:doc:'parrot'` occurs in the document `sketches/index`, then the link refers to `sketches/parrot`. If the reference is `:doc:'/people'` or `:doc:'../people'`, the link refers to `people`.

If no explicit link text is given (like usual: `:doc:'Monty Python members </people>'`), the link caption will be the title of the given document.

Referencing downloadable files

New in version 0.6.

:download:
This role lets you link to files within your source tree that are not reST documents that can be viewed, but files that can be downloaded.

When you use this role, the referenced file is automatically marked for inclusion in the output when building (obviously, for HTML output only). All downloadable files are put into the `_downloads` subdirectory of the output directory; duplicate filenames are handled.

An example:

```
See :download:`this example script <../example.py>`.
```

The given filename is usually relative to the directory the current source file is contained in, but if it absolute (starting with `/`), it is taken as relative to the top source directory.

The `example.py` file will be copied to the output directory, and a suitable link generated to it.

Cross-referencing figures by figure number

New in version 1.3.

:numref:
Link to the specified figures, tables and code-blocks; the standard reST labels are used. When you use this role, it will insert a reference to the figure with link text by its figure number like "Fig. 1.1".

[9] http://docutils.sourceforge.net/docs/ref/rst/directives.html#table

If an explicit link text is given (like usual: `:numref:'Image of Sphinx (Fig. %s) <my-figure>'`), the link caption will be the title of the reference. As a special character, *%s* will be replaced to figure number.

If *numfig* is `False`, figures are not numbered. so this role inserts not a reference but labels or link text.

Cross-referencing other items of interest

The following roles do possibly create a cross-reference, but do not refer to objects:

:envvar:
An environment variable. Index entries are generated. Also generates a link to the matching *envvar* directive, if it exists.

:token:
The name of a grammar token (used to create links between *productionlist* directives).

:keyword:
The name of a keyword in Python. This creates a link to a reference label with that name, if it exists.

:option:
A command-line option to an executable program. The leading hyphen(s) must be included. This generates a link to a *option* directive, if it exists.

The following role creates a cross-reference to a term in a *glossary*:

:term:
Reference to a term in a glossary. A glossary is created using the `glossary` directive containing a definition list with terms and definitions. It does not have to be in the same file as the `term` markup, for example the Python docs have one global glossary in the `glossary.rst` file.

If you use a term that's not explained in a glossary, you'll get a warning during build.

7.7.2 Other semantic markup

The following roles don't do anything special except formatting the text in a different style:

:abbr:
An abbreviation. If the role content contains a parenthesized explanation, it will be treated specially: it will be shown in a tool-tip in HTML, and output only once in LaTeX.

Example: `:abbr:'LIFO (last-in, first-out)'`.

New in version 0.6.

:command:
The name of an OS-level command, such as `rm`.

:dfn:
Mark the defining instance of a term in the text. (No index entries are generated.)

:file:
The name of a file or directory. Within the contents, you can use curly braces to indicate a "variable" part, for example:

```
... is installed in :file:`/usr/lib/python2.{x}/site-packages` ...
```

In the built documentation, the `x` will be displayed differently to indicate that it is to be replaced by the Python minor version.

:guilabel:
Labels presented as part of an interactive user interface should be marked using `guilabel`. This includes labels from text-based interfaces such as those created using `curses` or other text-based libraries. Any label used in the interface should be marked with this role, including button labels, window titles, field names, menu and menu selection names, and even values in selection lists.

Changed in version 1.0: An accelerator key for the GUI label can be included using an ampersand; this will be stripped and displayed underlined in the output (example: `:guilabel:'&Cancel'`). To include a literal ampersand, double it.

:kbd:
Mark a sequence of keystrokes. What form the key sequence takes may depend on platform- or application-specific conventions. When there are no relevant conventions, the names of modifier keys should be spelled out, to improve accessibility for new users and non-native speakers. For example, an *xemacs* key sequence may be marked like `:kbd:'C-x C-f'`, but without reference to a specific application or platform, the same sequence should be marked as `:kbd:'Control-x Control-f'`.

:mailheader:
The name of an RFC 822-style mail header. This markup does not imply that the header is being used in an email message, but can be used to refer to any header of the same "style." This is also used for headers defined by the various MIME specifications. The header name should be entered in the same way it would normally be found in practice, with the camel-casing conventions being preferred where there is more than one common usage. For example: `:mailheader:'Content-Type'`.

:makevar:
The name of a **make** variable.

:manpage:
A reference to a Unix manual page including the section, e.g. `:manpage:'ls(1)'`.

:menuselection:
Menu selections should be marked using the `menuselection` role. This is used to mark a complete sequence of menu selections, including selecting submenus and choosing a specific operation, or any subsequence of such a sequence. The names of individual selections should be separated by `-->`.

For example, to mark the selection "Start > Programs", use this markup:

```
:menuselection:`Start --> Programs`
```

When including a selection that includes some trailing indicator, such as the ellipsis some operating systems use to indicate that the command opens a dialog, the indicator should be omitted from the selection name.

`menuselection` also supports ampersand accelerators just like *`guilabel`*.

:mimetype:
The name of a MIME type, or a component of a MIME type (the major or minor portion, taken alone).

:newsgroup:
The name of a Usenet newsgroup.

:program:
The name of an executable program. This may differ from the file name for the executable for some platforms. In particular, the `.exe` (or other) extension should be omitted for Windows programs.

:regexp:
A regular expression. Quotes should not be included.

:samp:
A piece of literal text, such as code. Within the contents, you can use curly braces to indicate a "vari-

able" part, as in *file*. For example, in `:samp:`print 1+{variable}``, the part `variable` would be emphasized.

If you don't need the "variable part" indication, use the standard ``code`` instead.

There is also an *index* role to generate index entries.

The following roles generate external links:

:pep:
A reference to a Python Enhancement Proposal. This generates appropriate index entries. The text "PEP *number*" is generated; in the HTML output, this text is a hyperlink to an online copy of the specified PEP. You can link to a specific section by saying `:pep:`number#anchor``.

:rfc:
A reference to an Internet Request for Comments. This generates appropriate index entries. The text "RFC *number*" is generated; in the HTML output, this text is a hyperlink to an online copy of the specified RFC. You can link to a specific section by saying `:rfc:`number#anchor``.

Note that there are no special roles for including hyperlinks as you can use the standard reST markup for that purpose.

7.7.3 Substitutions

The documentation system provides three substitutions that are defined by default. They are set in the build configuration file.

|release|
Replaced by the project release the documentation refers to. This is meant to be the full version string including alpha/beta/release candidate tags, e.g. `2.5.2b3`. Set by *release*.

|version|
Replaced by the project version the documentation refers to. This is meant to consist only of the major and minor version parts, e.g. `2.5`, even for version 2.5.1. Set by *version*.

|today|
Replaced by either today's date (the date on which the document is read), or the date set in the build configuration file. Normally has the format `April 14, 2007`. Set by *today_fmt* and *today*.

7.8 Miscellaneous markup

7.8.1 File-wide metadata

reST has the concept of "field lists"; these are a sequence of fields marked up like this:

```
:fieldname: Field content
```

A field list near the top of a file is parsed by docutils as the "docinfo" which is normally used to record the author, date of publication and other metadata. *In Sphinx*, a field list preceding any other markup is moved from the docinfo to the Sphinx environment as document metadata and is not displayed in the output; a field list appearing after the document title will be part of the docinfo as normal and will be displayed in the output.

At the moment, these metadata fields are recognized:

tocdepth The maximum depth for a table of contents of this file.

New in version 0.4.

nocomments If set, the web application won't display a comment form for a page generated from this source file.

orphan If set, warnings about this file not being included in any toctree will be suppressed.

New in version 1.0.

7.8.2 Meta-information markup

.. **sectionauthor**:: name <email>
Identifies the author of the current section. The argument should include the author's name such that it can be used for presentation and email address. The domain name portion of the address should be lower case. Example:

```
.. sectionauthor:: Guido van Rossum <guido@python.org>
```

By default, this markup isn't reflected in the output in any way (it helps keep track of contributions), but you can set the configuration value *show_authors* to `True` to make them produce a paragraph in the output.

.. **codeauthor**:: name <email>
The *codeauthor* directive, which can appear multiple times, names the authors of the described code, just like *sectionauthor* names the author(s) of a piece of documentation. It too only produces output if the *show_authors* configuration value is `True`.

7.8.3 Index-generating markup

Sphinx automatically creates index entries from all object descriptions (like functions, classes or attributes) like discussed in *Sphinx Domains*.

However, there is also explicit markup available, to make the index more comprehensive and enable index entries in documents where information is not mainly contained in information units, such as the language reference.

.. **index**:: <entries>
This directive contains one or more index entries. Each entry consists of a type and a value, separated by a colon.

For example:

```
.. index::
   single: execution; context
   module: __main__
   module: sys
   triple: module; search; path

The execution context
---------------------

...
```

This directive contains five entries, which will be converted to entries in the generated index which link to the exact location of the index statement (or, in case of offline media, the corresponding page number).

Since index directives generate cross-reference targets at their location in the source, it makes sense to put them *before* the thing they refer to – e.g. a heading, as in the example above.

The possible entry types are:

single Creates a single index entry. Can be made a subentry by separating the subentry text with a semicolon (this notation is also used below to describe what entries are created).

pair `pair: loop; statement` is a shortcut that creates two index entries, namely `loop; statement` and `statement; loop`.

triple Likewise, `triple: module; search; path` is a shortcut that creates three index entries, which are `module; search path`, `search; path, module` and `path; module search`.

see `see: entry; other` creates an index entry that refers from `entry` to `other`.

seealso Like `see`, but inserts "see also" instead of "see".

module, keyword, operator, object, exception, statement, builtin These all create two index entries. For example, `module: hashlib` creates the entries `module; hashlib` and `hashlib; module`. (These are Python-specific and therefore deprecated.)

You can mark up "main" index entries by prefixing them with an exclamation mark. The references to "main" entries are emphasized in the generated index. For example, if two pages contain

```
.. index:: Python
```

and one page contains

```
.. index:: ! Python
```

then the backlink to the latter page is emphasized among the three backlinks.

For index directives containing only "single" entries, there is a shorthand notation:

```
.. index:: BNF, grammar, syntax, notation
```

This creates four index entries.

Changed in version 1.1: Added `see` and `seealso` types, as well as marking main entries.

:index:
While the *index* directive is a block-level markup and links to the beginning of the next paragraph, there is also a corresponding role that sets the link target directly where it is used.

The content of the role can be a simple phrase, which is then kept in the text and used as an index entry. It can also be a combination of text and index entry, styled like with explicit targets of cross-references. In that case, the "target" part can be a full entry as described for the directive above. For example:

```
This is a normal reST :index:`paragraph` that contains several
:index:`index entries <pair: index; entry>`.
```

New in version 1.1.

7.8.4 Including content based on tags

.. only:: <expression>
Include the content of the directive only if the *expression* is true. The expression should consist of tags, like this:

```
.. only:: html and draft
```

Undefined tags are false, defined tags (via the `-t` command-line option or within `conf.py`, see *here*) are true. Boolean expressions, also using parentheses (like `html and (latex or draft)`) are supported.

The *format* and the *name* of the current builder (`html`, `latex` or `text`) are always set as a tag [10]. To make the distinction between format and name explicit, they are also added with the prefix `format_` and `builder_`, e.g. the epub builder defines the tags `html`, `epub`, `format_html` and `builder_epub`.

These standard tags are set *after* the configuration file is read, so they are not available there.

All tags must follow the standard Python identifier syntax as set out in the Identifiers and keywords[11] documentation. That is, a tag expression may only consist of tags that conform to the syntax of Python variables. In ASCII, this consists of the uppercase and lowercase letters `A` through `Z`, the underscore _ and, except for the first character, the digits `0` through `9`.

New in version 0.6.

Changed in version 1.2: Added the name of the builder and the prefixes.

7.8.5 Tables

Use *standard reStructuredText tables*. They work fine in HTML output, however there are some gotchas when using tables in LaTeX: the column width is hard to determine correctly automatically. For this reason, the following directive exists:

`.. tabularcolumns::` `column spec`

This directive gives a "column spec" for the next table occurring in the source file. The spec is the second argument to the LaTeX `tabulary` package's environment (which Sphinx uses to translate tables). It can have values like

```
|l|l|l|
```

which means three left-adjusted, nonbreaking columns. For columns with longer text that should automatically be broken, use either the standard `p{width}` construct, or tabulary's automatic specifiers:

`L`	flush left column with automatic width
`R`	flush right column with automatic width
`C`	centered column with automatic width
`J`	justified column with automatic width

The automatic width is determined by rendering the content in the table, and scaling them according to their share of the total width.

By default, Sphinx uses a table layout with `L` for every column.

New in version 0.3.

[10] For most builders name and format are the same. At the moment only builders derived from the html builder distinguish between the builder format and the builder name.
Note that the current builder tag is not available in `conf.py`, it is only available after the builder is initialized.
[11] https://docs.python.org/reference/lexical_analysis.html#identifiers

Warning: Tables that contain list-like elements such as object descriptions, blockquotes or any kind of lists cannot be set out of the box with `tabulary`. They are therefore set with the standard LaTeX `tabular` environment if you don't give a `tabularcolumns` directive. If you do, the table will be set with `tabulary`, but you must use the `p{width}` construct for the columns that contain these elements. Literal blocks do not work with `tabulary` at all, so tables containing a literal block are always set with `tabular`. Also, the verbatim environment used for literal blocks only works in `p{width}` columns, which means that by default, Sphinx generates such column specs for such tables. Use the *`tabularcolumns`* directive to get finer control over such tables.

More markup is added by *Sphinx Domains*.

CHAPTER 8

Sphinx Domains

New in version 1.0.

8.1 What is a Domain?

Originally, Sphinx was conceived for a single project, the documentation of the Python language. Shortly afterwards, it was made available for everyone as a documentation tool, but the documentation of Python modules remained deeply built in – the most fundamental directives, like `function`, were designed for Python objects. Since Sphinx has become somewhat popular, interest developed in using it for many different purposes: C/C++ projects, JavaScript, or even reStructuredText markup (like in this documentation).

While this was always possible, it is now much easier to easily support documentation of projects using different programming languages or even ones not supported by the main Sphinx distribution, by providing a **domain** for every such purpose.

A domain is a collection of markup (reStructuredText *directive*s and *role*s) to describe and link to *object*s belonging together, e.g. elements of a programming language. Directive and role names in a domain have names like `domain:name`, e.g. `py:function`. Domains can also provide custom indices (like the Python Module Index).

Having domains means that there are no naming problems when one set of documentation wants to refer to e.g. C++ and Python classes. It also means that extensions that support the documentation of whole new languages are much easier to write.

This section describes what the domains that come with Sphinx provide. The domain API is documented as well, in the section *Domain API*.

8.2 Basic Markup

Most domains provide a number of *object description directives*, used to describe specific objects provided by modules. Each directive requires one or more signatures to provide basic information about what is being described, and the content should be the description. The basic version makes entries in the general index; if no index entry is desired, you can give the directive option flag `:noindex:`. An example using a Python domain directive:

```
.. py:function:: spam(eggs)
                 ham(eggs)

   Spam or ham the foo.
```

This describes the two Python functions `spam` and `ham`. (Note that when signatures become too long, you can break them if you add a backslash to lines that are continued in the next line. Example:

```
.. py:function:: filterwarnings(action, message='', category=Warning, \
                                 module='', lineno=0, append=False)
   :noindex:
```

(This example also shows how to use the `:noindex:` flag.)

The domains also provide roles that link back to these object descriptions. For example, to link to one of the functions described in the example above, you could say

```
The function :py:func:`spam` does a similar thing.
```

As you can see, both directive and role names contain the domain name and the directive name.

Default Domain

To avoid having to writing the domain name all the time when you e.g. only describe Python objects, a default domain can be selected with either the config value *primary_domain* or this directive:

.. default-domain:: name
Select a new default domain. While the *primary_domain* selects a global default, this only has an effect within the same file.

If no other default is selected, the Python domain (named `py`) is the default one, mostly for compatibility with documentation written for older versions of Sphinx.

Directives and roles that belong to the default domain can be mentioned without giving the domain name, i.e.

```
.. function:: pyfunc()

   Describes a Python function.

Reference to :func:`pyfunc`.
```

8.2.1 Cross-referencing syntax

For cross-reference roles provided by domains, the same facilities exist as for general cross-references. See *Cross-referencing syntax*.

In short:

- You may supply an explicit title and reference target: `:role:`title <target>`` will refer to *target*, but the link text will be *title*.
- If you prefix the content with `!`, no reference/hyperlink will be created.
- If you prefix the content with `~`, the link text will only be the last component of the target. For example, `:py:meth:`~Queue.Queue.get`` will refer to `Queue.Queue.get` but only display `get` as the link text.

8.3 The Python Domain

The Python domain (name **py**) provides the following directives for module declarations:

.. **py:module::** name
This directive marks the beginning of the description of a module (or package submodule, in which case the name should be fully qualified, including the package name). It does not create content (like e.g. *py:class* does).

This directive will also cause an entry in the global module index.

The `platform` option, if present, is a comma-separated list of the platforms on which the module is available (if it is available on all platforms, the option should be omitted). The keys are short identifiers; examples that are in use include "IRIX", "Mac", "Windows", and "Unix". It is important to use a key which has already been used when applicable.

The `synopsis` option should consist of one sentence describing the module's purpose – it is currently only used in the Global Module Index.

The `deprecated` option can be given (with no value) to mark a module as deprecated; it will be designated as such in various locations then.

.. **py:currentmodule::** name
This directive tells Sphinx that the classes, functions etc. documented from here are in the given module (like *py:module*), but it will not create index entries, an entry in the Global Module Index, or a link target for *py:mod*. This is helpful in situations where documentation for things in a module is spread over multiple files or sections – one location has the *py:module* directive, the others only *py:currentmodule*.

The following directives are provided for module and class contents:

.. **py:function::** name(parameters)
Describes a module-level function. The signature should include the parameters as given in the Python function definition, see *Python Signatures*. For example:

```
.. py:function:: Timer.repeat(repeat=3, number=1000000)
```

For methods you should use *py:method*.

The description normally includes information about the parameters required and how they are used (especially whether mutable objects passed as parameters are modified), side effects, and possible exceptions.

This information can (in any `py` directive) optionally be given in a structured form, see *Info field lists*.

.. **py:data::** name
Describes global data in a module, including both variables and values used as "defined constants." Class and object attributes are not documented using this environment.

.. **py:exception::** name
Describes an exception class. The signature can, but need not include parentheses with constructor arguments.

.. **py:class::** name
.. **py:class::** name(parameters)
Describes a class. The signature can optionally include parentheses with parameters which will be shown as the constructor arguments. See also *Python Signatures*.

Methods and attributes belonging to the class should be placed in this directive's body. If they are placed outside, the supplied name should contain the class name so that cross-references still work. Example:

```
.. py:class:: Foo

   .. py:method:: quux()
```

```
    -- or --

    .. py:class:: Bar

    .. py:method:: Bar.quux()
```

The first way is the preferred one.

.. **py:attribute::** name
Describes an object data attribute. The description should include information about the type of the data to be expected and whether it may be changed directly.

.. **py:method::** name(parameters)
Describes an object method. The parameters should not include the `self` parameter. The description should include similar information to that described for `function`. See also *Python Signatures* and *Info field lists*.

.. **py:staticmethod::** name(parameters)
Like *py:method*, but indicates that the method is a static method.

New in version 0.4.

.. **py:classmethod::** name(parameters)
Like *py:method*, but indicates that the method is a class method.

New in version 0.6.

.. **py:decorator::** name
.. **py:decorator::** name(parameters)
Describes a decorator function. The signature should represent the usage as a decorator. For example, given the functions

```
    def removename(func):
        func.__name__ = ''
        return func

    def setnewname(name):
        def decorator(func):
            func.__name__ = name
            return func
        return decorator
```

the descriptions should look like this:

```
    .. py:decorator:: removename

       Remove name of the decorated function.

    .. py:decorator:: setnewname(name)

       Set name of the decorated function to *name*.
```

(as opposed to `.. py:decorator:: removename(func)`.)

There is no `py:deco` role to link to a decorator that is marked up with this directive; rather, use the *py:func* role.

.. **py:decoratormethod::** name
.. **py:decoratormethod::** name(signature)
Same as *py:decorator*, but for decorators that are methods.

Refer to a decorator method using the `py:meth` role.

8.3.1 Python Signatures

Signatures of functions, methods and class constructors can be given like they would be written in Python.

Default values for optional arguments can be given (but if they contain commas, they will confuse the signature parser). Python 3-style argument annotations can also be given as well as return type annotations:

```
.. py:function:: compile(source : string, filename, symbol='file') -> ast object
```

For functions with optional parameters that don't have default values (typically functions implemented in C extension modules without keyword argument support), you can use brackets to specify the optional parts:

compile(*source*[, *filename*[, *symbol*]])

It is customary to put the opening bracket before the comma.

8.3.2 Info field lists

New in version 0.4.

Inside Python object description directives, reST field lists with these fields are recognized and formatted nicely:

- `param`, `parameter`, `arg`, `argument`, `key`, `keyword`: Description of a parameter.
- `type`: Type of a parameter. Creates a link if possible.
- `raises`, `raise`, `except`, `exception`: That (and when) a specific exception is raised.
- `var`, `ivar`, `cvar`: Description of a variable.
- `vartype`: Type of a variable. Creates a link if possible.
- `returns`, `return`: Description of the return value.
- `rtype`: Return type. Creates a link if possible.

The field names must consist of one of these keywords and an argument (except for `returns` and `rtype`, which do not need an argument). This is best explained by an example:

```
.. py:function:: send_message(sender, recipient, message_body, [priority=1])

   Send a message to a recipient

   :param str sender: The person sending the message
   :param str recipient: The recipient of the message
   :param str message_body: The body of the message
   :param priority: The priority of the message, can be a number 1-5
   :type priority: integer or None
   :return: the message id
   :rtype: int
   :raises ValueError: if the message_body exceeds 160 characters
   :raises TypeError: if the message_body is not a basestring
```

This will render like this:

send_message(*sender*, *recipient*, *message_body*[, *priority=1*])

Send a message to a recipient

Parameters

- **sender** (*str*) – The person sending the message
- **recipient** (*str*) – The recipient of the message
- **message_body** (*str*) – The body of the message
- **priority** (*integer or None*) – The priority of the message, can be a number 1-5

Returns the message id

Return type int

Raises

- **ValueError** – if the message_body exceeds 160 characters
- **TypeError** – if the message_body is not a basestring

It is also possible to combine parameter type and description, if the type is a single word, like this:

```
:param int priority: The priority of the message, can be a number 1-5
```

8.3.3 Cross-referencing Python objects

The following roles refer to objects in modules and are possibly hyperlinked if a matching identifier is found:

:py:mod:
Reference a module; a dotted name may be used. This should also be used for package names.

:py:func:
Reference a Python function; dotted names may be used. The role text needs not include trailing parentheses to enhance readability; they will be added automatically by Sphinx if the *add_function_parentheses* config value is `True` (the default).

:py:data:
Reference a module-level variable.

:py:const:
Reference a "defined" constant. This may be a Python variable that is not intended to be changed.

:py:class:
Reference a class; a dotted name may be used.

:py:meth:
Reference a method of an object. The role text can include the type name and the method name; if it occurs within the description of a type, the type name can be omitted. A dotted name may be used.

:py:attr:
Reference a data attribute of an object.

:py:exc:
Reference an exception. A dotted name may be used.

:py:obj:
Reference an object of unspecified type. Useful e.g. as the *default_role*.

New in version 0.4.

The name enclosed in this markup can include a module name and/or a class name. For example, `:py:func:`filter`` could refer to a function named `filter` in the current module, or the built-in function of that name. In contrast, `:py:func:`foo.filter`` clearly refers to the `filter` function in the `foo` module.

Normally, names in these roles are searched first without any further qualification, then with the current module name prepended, then with the current module and class name (if any) prepended. If you prefix the name with a dot, this order is reversed. For example, in the documentation of Python's `codecs` module, `:py:func:`open`` always refers to the built-in function, while `:py:func:`.open`` refers to `codecs.open()`.

A similar heuristic is used to determine whether the name is an attribute of the currently documented class.

Also, if the name is prefixed with a dot, and no exact match is found, the target is taken as a suffix and all object names with that suffix are searched. For example, `:py:meth:`.TarFile.close`` references the `tarfile.TarFile.close()` function, even if the current module is not `tarfile`. Since this can get ambiguous, if there is more than one possible match, you will get a warning from Sphinx.

Note that you can combine the `~` and `.` prefixes: `:py:meth:`~.TarFile.close`` will reference the `tarfile.TarFile.close()` method, but the visible link caption will only be `close()`.

8.4 The C Domain

The C domain (name **c**) is suited for documentation of C API.

`.. c:function:: type name(signature)`
Describes a C function. The signature should be given as in C, e.g.:

```
.. c:function:: PyObject* PyType_GenericAlloc(PyTypeObject *type, Py_ssize_t nitems)
```

This is also used to describe function-like preprocessor macros. The names of the arguments should be given so they may be used in the description.

Note that you don't have to backslash-escape asterisks in the signature, as it is not parsed by the reST inliner.

`.. c:member:: type name`
Describes a C struct member. Example signature:

```
.. c:member:: PyObject* PyTypeObject.tp_bases
```

The text of the description should include the range of values allowed, how the value should be interpreted, and whether the value can be changed. References to structure members in text should use the `member` role.

`.. c:macro:: name`
Describes a "simple" C macro. Simple macros are macros which are used for code expansion, but which do not take arguments so cannot be described as functions. This is a simple C-language `#define`. Examples of its use in the Python documentation include `PyObject_HEAD` and `Py_BEGIN_ALLOW_THREADS`.

`.. c:type:: name`
Describes a C type (whether defined by a typedef or struct). The signature should just be the type name.

`.. c:var:: type name`
Describes a global C variable. The signature should include the type, such as:

```
.. c:var:: PyObject* PyClass_Type
```

8.4.1 Cross-referencing C constructs

The following roles create cross-references to C-language constructs if they are defined in the documentation:

:c:data:
Reference a C-language variable.

:c:func:
Reference a C-language function. Should include trailing parentheses.

:c:macro:
Reference a "simple" C macro, as defined above.

:c:type:
Reference a C-language type.

8.5 The C++ Domain

The C++ domain (name **cpp**) supports documenting C++ projects.

The following directives are available. All declarations can start with a visibility statement (`public`, `private` or `protected`).

.. cpp:class:: class specifier
Describe a class/struct, possibly with specification of inheritance, e.g.,:

```
.. cpp:class:: MyClass : public MyBase, MyOtherBase
```

The class can be directly declared inside a nested scope, e.g.,:

```
.. cpp:class:: OuterScope::MyClass : public MyBase, MyOtherBase
```

A template class can be declared:

```
.. cpp:class:: template<typename T, std::size_t N> std::array
```

or with a line beak:

```
.. cpp:class:: template<typename T, std::size_t N> \
               std::array
```

Full and partial template specialisations can be declared:

```
.. cpp::class:: template<> \
                std::array<bool, 256>

.. cpp::class:: template<typename T> \
                std::array<T, 42>
```

.. cpp:function:: (member) function prototype
Describe a function or member function, e.g.,:

```
.. cpp:function:: bool myMethod(int arg1, std::string arg2)

   A function with parameters and types.

.. cpp:function:: bool myMethod(int, double)

   A function with unnamed parameters.

.. cpp:function:: const T &MyClass::operator[](std::size_t i) const

   An overload for the indexing operator.

.. cpp:function:: operator bool() const

   A casting operator.

.. cpp:function:: constexpr void foo(std::string &bar[2]) noexcept

   A constexpr function.

.. cpp:function:: MyClass::MyClass(const MyClass&) = default

   A copy constructor with default implementation.
```

Function templates can also be described:

```
.. cpp:function:: template<typename U> \
                  void print(U &&u)
```

and function template specialisations:

```
.. cpp:function:: template<> \
                  void print(int i)
```

.. cpp:member:: (member) variable declaration
.. cpp:var:: (member) variable declaration

Describe a varible or member variable, e.g.,:

```
.. cpp:member:: std::string MyClass::myMember

.. cpp:var:: std::string MyClass::myOtherMember[N][M]

.. cpp:member:: int a = 42
```

Variable templates can also be described:

```
.. cpp:member:: template<class T> \
                constexpr T pi = T(3.1415926535897932385)
```

.. cpp:type:: typedef declaration
.. cpp:type:: name
.. cpp:type:: type alias declaration

Describe a type as in a typedef declaration, a type alias declaration, or simply the name of a type with unspecified type, e.g.,:

```
.. cpp:type:: std::vector<int> MyList

   A typedef-like declaration of a type.
```

```
.. cpp:type:: MyContainer::const_iterator

   Declaration of a type alias with unspecified type.

.. cpp:type:: MyType = std::unordered_map<int, std::string>

   Declaration of a type alias.
```

A type alias can also be templated:

```
.. cpp:type:: template<typename T>
              MyContainer = std::vector<T>
```

.. **cpp:enum::** unscoped enum declaration
.. **cpp:enum-struct::** scoped enum declaration
.. **cpp:enum-class::** scoped enum declaration

Describe a (scoped) enum, possibly with the underlying type specified. Any enumerators declared inside an unscoped enum will be declared both in the enum scope and in the parent scope. Examples:

```
.. cpp:enum:: MyEnum

   An unscoped enum.

.. cpp:enum:: MySpecificEnum : long

   An unscoped enum with specified underlying type.

.. cpp:enum-class:: MyScopedEnum

   A scoped enum.

.. cpp:enum-struct:: protected MyScopedVisibilityEnum : std::underlying_type<MySpecificEnum>::ty

   A scoped enum with non-default visibility, and with a specified underlying type.
```

.. **cpp:enumerator::** name
.. **cpp:enumerator::** name = constant

Describe an enumerator, optionally with its value defined, e.g.,:

```
.. cpp::enumerator:: MyEnum::myEnumerator

.. cpp::enumerator:: MyEnum::myOtherEnumerator = 42
```

8.5.1 Namespacing

Declarations in the C++ doamin are as default placed in global scope. The current scope can be changed using three namespace directives. They manage a stack declarations where `cpp:namespace` resets the stack and changes a given scope. The `cpp:namespace-push` directive changes the scope to a given inner scope of the current one. The `cpp:namespace-pop` directive undos the most recent `cpp:namespace-push` directive.

.. **cpp:namespace::** scope specification

Changes the current scope for the subsequent objects to the given scope, and resets the namespace directive stack. Note that the namespace does not need to correspond to C++ namespaces, but can end in names of classes, e.g.,:

```
.. cpp:namespace:: Namespace1::Namespace2::SomeClass::AnInnerClass
```

All subsequent objects will be defined as if their name were declared with the scope prepended. The subsequent cross-references will be searched for starting in the current scope.

Using `NULL`, `0`, or `nullptr` as the scope will change to global scope.

A namespace declaration can also be templated, e.g.,:

```
.. cpp:class:: template<typename T> \
               std::vector

.. cpp:namespace:: template<typename T> std::vector

.. cpp:function:: std::size_t size() const
```

declares `size` as a member function of the template class `std::vector`. Equivalently this could have been declared using:

```
.. cpp:class:: template<typename T> \
               std::vector

   .. cpp:function:: std::size_t size() const
```

or::

```
.. cpp:class:: template<typename T> \
               std::vector
```

.. **cpp:namespace-push::** `scope specification`
Change the scope relatively to the current scope. For example, after:

```
.. cpp:namespace:: A::B

.. cpp:namespace-push:: C::D
```

the current scope will be `A::B::C::D`.

.. **cpp:namespace-pop::**
Undo the previous `cpp:namespace-push` directive (*not* just pop a scope). For example, after:

```
.. cpp:namespace:: A::B

.. cpp:namespace-push:: C::D

.. cpp:namespace-pop::
```

the current scope will be `A::B` (*not* `A::B::C`).

If no previous `cpp:namespace-push` directive has been used, but only a `cpp:namespace` directive, then the current scope will be reset to global scope. That is, `.. cpp:namespace:: A::B` is equivalent to:

```
.. cpp:namespace:: nullptr

.. cpp:namespace-push:: A::B
```

8.5.2 Info field lists

The C++ directives support the following info fields (see also *Info field lists*):

- *param*, *parameter*, *arg*, *argument*: Description of a parameter.
- *returns*, *return*: Description of a return value.
- *throws*, *throw*, *exception*: Description of a possibly thrown exception.

8.5.3 Cross-referencing

These roles link to the given object types:

:cpp:any:
:cpp:class:
:cpp:func:
:cpp:member:
:cpp:var:
:cpp:type:
:cpp:enum:
:cpp:enumerator:
Reference a C++ object by name. The name must be properly qualified relative to the position of the link.

Note: Sphinx's syntax to give references a custom title can interfere with linking to template classes, if nothing follows the closing angle bracket, i.e. if the link looks like this: `:cpp:class:`MyClass<T>``. This is interpreted as a link to `T` with a title of `MyClass`. In this case, please escape the opening angle bracket with a backslash, like this: `:cpp:class:`MyClass\<T>``.

Note on References

It is currently impossible to link to a specific version of an overloaded method. Currently the C++ domain is the first domain that has basic support for overloaded methods and until there is more data for comparison we don't want to select a bad syntax to reference a specific overload. Currently Sphinx will link to the first overloaded version of the method / function.

8.6 The Standard Domain

The so-called "standard" domain collects all markup that doesn't warrant a domain of its own. Its directives and roles are not prefixed with a domain name.

The standard domain is also where custom object descriptions, added using the *add_object_type()* API, are placed.

There is a set of directives allowing documenting command-line programs:

.. option:: name args, name args, ...
Describes a command line argument or switch. Option argument names should be enclosed in angle brackets. Examples:

```
.. option:: dest_dir

   Destination directory.

.. option:: -m <module>, --module <module>

   Run a module as a script.
```

The directive will create cross-reference targets for the given options, referencable by *option* (in the example case, you'd use something like `:option:'dest_dir'`, `:option:'-m'`, or `:option:'--module'`).

`cmdoption` directive is a deprecated alias for the `option` directive.

`.. envvar:: name`
Describes an environment variable that the documented code or program uses or defines. Referencable by *envvar*.

`.. program:: name`
Like *py:currentmodule*, this directive produces no output. Instead, it serves to notify Sphinx that all following *option* directives document options for the program called *name*.

If you use *program*, you have to qualify the references in your *option* roles by the program name, so if you have the following situation

```
.. program:: rm

.. option:: -r

   Work recursively.

.. program:: svn

.. option:: -r revision

   Specify the revision to work upon.
```

then `:option:'rm -r'` would refer to the first option, while `:option:'svn -r'` would refer to the second one.

The program name may contain spaces (in case you want to document subcommands like `svn add` and `svn commit` separately).

New in version 0.5.

There is also a very generic object description directive, which is not tied to any domain:

`.. describe:: text`
`.. object:: text`
This directive produces the same formatting as the specific ones provided by domains, but does not create index entries or cross-referencing targets. Example:

```
.. describe:: PAPER

   You can set this variable to select a paper size.
```

8.7 The JavaScript Domain

The JavaScript domain (name **js**) provides the following directives:

`.. js:function:: name(signature)`
Describes a JavaScript function or method. If you want to describe arguments as optional use square brackets as *documented* for Python signatures.

You can use fields to give more details about arguments and their expected types, errors which may be thrown by the function, and the value being returned:

```
.. js:function:: $.getJSON(href, callback[, errback])

   :param string href: An URI to the location of the resource.
   :param callback: Gets called with the object.
   :param errback:
       Gets called in case the request fails. And a lot of other
       text so we need multiple lines.
   :throws SomeError: For whatever reason in that case.
   :returns: Something.
```

This is rendered as:

$.**getJSON**(*href*, *callback*[, *errback*])

Arguments
- **href** (*string*) – An URI to the location of the resource.
- **callback** – Gets called with the object.
- **errback** – Gets called in case the request fails. And a lot of other text so we need multiple lines.

Throws SomeError For whatever reason in that case.

Returns Something.

`.. js:class:: name`
Describes a constructor that creates an object. This is basically like a function but will show up with a *class* prefix:

```
.. js:class:: MyAnimal(name[, age])

   :param string name: The name of the animal
   :param number age: an optional age for the animal
```

This is rendered as:

class **MyAnimal**(*name*[, *age*])

Arguments
- **name** (*string*) – The name of the animal
- **age** (*number*) – an optional age for the animal

`.. js:data:: name`
Describes a global variable or constant.

`.. js:attribute:: object.name`
Describes the attribute *name* of *object*.

These roles are provided to refer to the described objects:

:js:func:
:js:class:
:js:data:
:js:attr:

8.8 The reStructuredText domain

The reStructuredText domain (name **rst**) provides the following directives:

`.. rst:directive:: name`
Describes a reST directive. The *name* can be a single directive name or actual directive syntax (.. prefix and :: suffix) with arguments that will be rendered differently. For example:

```
.. rst:directive:: foo

   Foo description.

.. rst:directive:: .. bar:: baz

   Bar description.
```

will be rendered as:

> **.. foo::**
> Foo description.
>
> **.. bar::** baz
> Bar description.

.. rst:role:: name
Describes a reST role. For example:

```
.. rst:role:: foo

   Foo description.
```

will be rendered as:

> **:foo:**
> Foo description.

These roles are provided to refer to the described objects:

:rst:dir:
:rst:role:

8.9 More domains

The sphinx-contrib[1] repository contains more domains available as extensions; currently Ada[2], CoffeeScript[3], Erlang[4], HTTP[5], Lasso[6], MATLAB[7], PHP[8], and Ruby[9] domains. Also available are domains for Chapel[10], Common Lisp[11], dqn[12], Go[13], Jinja[14], Operation[15], and Scala[16].

[1] https://bitbucket.org/birkenfeld/sphinx-contrib/
[2] https://pypi.python.org/pypi/sphinxcontrib-adadomain
[3] https://pypi.python.org/pypi/sphinxcontrib-coffee
[4] https://pypi.python.org/pypi/sphinxcontrib-erlangdomain
[5] https://pypi.python.org/pypi/sphinxcontrib-httpdomain
[6] https://pypi.python.org/pypi/sphinxcontrib-lassodomain
[7] https://pypi.python.org/pypi/sphinxcontrib-matlabdomain
[8] https://pypi.python.org/pypi/sphinxcontrib-phpdomain
[9] https://bitbucket.org/birkenfeld/sphinx-contrib/src/default/rubydomain
[10] https://pypi.python.org/pypi/sphinxcontrib-chapeldomain
[11] https://pypi.python.org/pypi/sphinxcontrib-cldomain
[12] https://pypi.python.org/pypi/sphinxcontrib-dqndomain
[13] https://pypi.python.org/pypi/sphinxcontrib-golangdomain
[14] https://pypi.python.org/pypi/sphinxcontrib-jinjadomain
[15] https://pypi.python.org/pypi/sphinxcontrib-operationdomain
[16] https://pypi.python.org/pypi/sphinxcontrib-scaladomain

CHAPTER 9

Available builders

These are the built-in Sphinx builders. More builders can be added by *extensions*.

The builder's "name" must be given to the **-b** command-line option of **sphinx-build** to select a builder.

class sphinx.builders.html.**StandaloneHTMLBuilder**
This is the standard HTML builder. Its output is a directory with HTML files, complete with style sheets and optionally the reST sources. There are quite a few configuration values that customize the output of this builder, see the chapter *Options for HTML output* for details.

name = 'html'

supported_image_types = ['image/svg+xml', 'image/png', 'image/gif', 'image/jpeg']

class sphinx.builders.html.**DirectoryHTMLBuilder**
This is a subclass of the standard HTML builder. Its output is a directory with HTML files, where each file is called `index.html` and placed in a subdirectory named like its page name. For example, the document `markup/rest.rst` will not result in an output file `markup/rest.html`, but `markup/rest/index.html`. When generating links between pages, the `index.html` is omitted, so that the URL would look like `markup/rest/`.

name = 'dirhtml'

supported_image_types = ['image/svg+xml', 'image/png', 'image/gif', 'image/jpeg']

New in version 0.6.

class sphinx.builders.html.**SingleFileHTMLBuilder**
This is an HTML builder that combines the whole project in one output file. (Obviously this only works with smaller projects.) The file is named like the master document. No indices will be generated.

name = 'singlehtml'

supported_image_types = ['image/svg+xml', 'image/png', 'image/gif', 'image/jpeg']

New in version 1.0.

class sphinx.builders.htmlhelp.**HTMLHelpBuilder**
This builder produces the same output as the standalone HTML builder, but also generates HTML Help support files that allow the Microsoft HTML Help Workshop to compile them into a CHM file.

name = 'htmlhelp'

supported_image_types = ['image/png', 'image/gif', 'image/jpeg']

class sphinx.builders.qthelp.**QtHelpBuilder**
This builder produces the same output as the standalone HTML builder, but also generates Qt help[1] collection support files that allow the Qt collection generator to compile them.

name = 'qthelp'

supported_image_types = ['image/svg+xml', 'image/png', 'image/gif', 'image/jpeg']

class sphinx.builders.applehelp.**AppleHelpBuilder**
This builder produces an Apple Help Book based on the same output as the standalone HTML builder.

If the source directory contains any `.lproj` folders, the one corresponding to the selected language will have its contents merged with the generated output. These folders will be ignored by all other documentation types.

In order to generate a valid help book, this builder requires the command line tool **hiutil**, which is only available on Mac OS X 10.6 and above. You can disable the indexing step by setting *applehelp_disable_external_tools* to `True`, in which case the output will not be valid until **hiutil** has been run on all of the `.lproj` folders within the bundle.

name = 'applehelp'

supported_image_types = ['image/png', 'image/gif', 'image/jpeg', 'image/tiff', 'image/jp2', 'image/svg+xml']

New in version 1.3.

class sphinx.builders.devhelp.**DevhelpBuilder**
This builder produces the same output as the standalone HTML builder, but also generates GNOME Devhelp[2] support file that allows the GNOME Devhelp reader to view them.

name = 'devhelp'

supported_image_types = ['image/png', 'image/gif', 'image/jpeg']

class sphinx.builders.epub.**EpubBuilder**
This builder produces the same output as the standalone HTML builder, but also generates an *epub* file for ebook readers. See *Epub info* for details about it. For definition of the epub format, have a look at http://idpf.org/epub or http://en.wikipedia.org/wiki/EPUB. The builder creates *EPUB 2* files.

name = 'epub'

supported_image_types = ['image/svg+xml', 'image/png', 'image/gif', 'image/jpeg']

class sphinx.builders.latex.**LaTeXBuilder**
This builder produces a bunch of LaTeX files in the output directory. You have to specify which documents are to be included in which LaTeX files via the *latex_documents* configuration value. There are a few configuration values that customize the output of this builder, see the chapter *Options for LaTeX output* for details.

Note: The produced LaTeX file uses several LaTeX packages that may not be present in a "minimal" TeX distribution installation. For TeXLive, the following packages need to be installed:

- latex-recommended
- latex-extra
- fonts-recommended

---

name = 'latex'

[1] http://qt-project.org/doc/qt-4.8/qthelp-framework.html
[2] https://wiki.gnome.org/Apps/Devhelp

supported_image_types = ['application/pdf', 'image/png', 'image/gif', 'image/jpeg']

Note that a direct PDF builder using ReportLab is available in rst2pdf[3] version 0.12 or greater. You need to add `'rst2pdf.pdfbuilder'` to your *extensions* to enable it, its name is `pdf`. Refer to the rst2pdf manual[4] for details.

class sphinx.builders.text.**TextBuilder**

This builder produces a text file for each reST file – this is almost the same as the reST source, but with much of the markup stripped for better readability.

name = 'text'

supported_image_types = []

New in version 0.4.

class sphinx.builders.manpage.**ManualPageBuilder**

This builder produces manual pages in the groff format. You have to specify which documents are to be included in which manual pages via the *man_pages* configuration value.

name = 'man'

supported_image_types = []

New in version 1.0.

class sphinx.builders.texinfo.**TexinfoBuilder**

This builder produces Texinfo files that can be processed into Info files by the **makeinfo** program. You have to specify which documents are to be included in which Texinfo files via the *texinfo_documents* configuration value.

The Info format is the basis of the on-line help system used by GNU Emacs and the terminal-based program **info**. See *Texinfo info* for more details. The Texinfo format is the official documentation system used by the GNU project. More information on Texinfo can be found at http://www.gnu.org/software/texinfo/.

name = 'texinfo'

supported_image_types = ['image/png', 'image/jpeg', 'image/gif']

New in version 1.1.

class sphinx.builders.html.**SerializingHTMLBuilder**

This builder uses a module that implements the Python serialization API (*pickle*, *simplejson*, *phpserialize*, and others) to dump the generated HTML documentation. The pickle builder is a subclass of it.

A concrete subclass of this builder serializing to the PHP serialization[5] format could look like this:

```
import phpserialize

class PHPSerializedBuilder(SerializingHTMLBuilder):
    name = 'phpserialized'
    implementation = phpserialize
    out_suffix = '.file.phpdump'
    globalcontext_filename = 'globalcontext.phpdump'
    searchindex_filename = 'searchindex.phpdump'
```

[3] http://rst2pdf.googlecode.com
[4] http://ralsina.me/static/manual.pdf
[5] https://pypi.python.org/pypi/phpserialize

implementation
A module that implements *dump()*, *load()*, *dumps()* and *loads()* functions that conform to the functions with the same names from the pickle module. Known modules implementing this interface are *simplejson* (or *json* in Python 2.6), *phpserialize*, *plistlib*, and others.

out_suffix
The suffix for all regular files.

globalcontext_filename
The filename for the file that contains the "global context". This is a dict with some general configuration values such as the name of the project.

searchindex_filename
The filename for the search index Sphinx generates.

See *Serialization builder details* for details about the output format.

New in version 0.5.

class sphinx.builders.html.**PickleHTMLBuilder**
This builder produces a directory with pickle files containing mostly HTML fragments and TOC information, for use of a web application (or custom postprocessing tool) that doesn't use the standard HTML templates.

See *Serialization builder details* for details about the output format.

name = 'pickle'
The old name `web` still works as well.

supported_image_types = ['image/svg+xml', 'image/png', 'image/gif', 'image/jpeg']

The file suffix is `.fpickle`. The global context is called `globalcontext.pickle`, the search index `searchindex.pickle`.

class sphinx.builders.html.**JSONHTMLBuilder**
This builder produces a directory with JSON files containing mostly HTML fragments and TOC information, for use of a web application (or custom postprocessing tool) that doesn't use the standard HTML templates.

See *Serialization builder details* for details about the output format.

name = 'json'

supported_image_types = ['image/svg+xml', 'image/png', 'image/gif', 'image/jpeg']

The file suffix is `.fjson`. The global context is called `globalcontext.json`, the search index `searchindex.json`.

New in version 0.5.

class sphinx.builders.gettext.**MessageCatalogBuilder**
This builder produces gettext-style message catalogs. Each top-level file or subdirectory grows a single `.pot` catalog template.

See the documentation on *Internationalization* for further reference.

name = u'gettext'

supported_image_types = []

New in version 1.1.

class sphinx.builders.changes.**ChangesBuilder**
This builder produces an HTML overview of all *versionadded*, *versionchanged* and

deprecated directives for the current *version*. This is useful to generate a ChangeLog file, for example.

name = 'changes'

supported_image_types = []

class sphinx.builders.linkcheck.**CheckExternalLinksBuilder**
This builder scans all documents for external links, tries to open them with `urllib2`, and writes an overview which ones are broken and redirected to standard output and to `output.txt` in the output directory.

name = 'linkcheck'

supported_image_types = []

class sphinx.builders.xml.**XMLBuilder**
This builder produces Docutils-native XML files. The output can be transformed with standard XML tools such as XSLT processors into arbitrary final forms.

name = 'xml'

supported_image_types = []

New in version 1.2.

class sphinx.builders.xml.**PseudoXMLBuilder**
This builder is used for debugging the Sphinx/Docutils "Reader to Transform to Writer" pipeline. It produces compact pretty-printed "pseudo-XML", files where nesting is indicated by indentation (no end-tags). External attributes for all elements are output, and internal attributes for any leftover "pending" elements are also given.

name = 'pseudoxml'

supported_image_types = []

New in version 1.2.

Built-in Sphinx extensions that offer more builders are:

- *doctest*
- *coverage*

9.1 Serialization builder details

All serialization builders outputs one file per source file and a few special files. They also copy the reST source files in the directory `_sources` under the output directory.

The *PickleHTMLBuilder* is a builtin subclass that implements the pickle serialization interface.

The files per source file have the extensions of *out_suffix*, and are arranged in directories just as the source files are. They unserialize to a dictionary (or dictionary like structure) with these keys:

body The HTML "body" (that is, the HTML rendering of the source file), as rendered by the HTML translator.

title The title of the document, as HTML (may contain markup).

toc The table of contents for the file, rendered as an HTML `<ul>`.

display_toc A boolean that is `True` if the `toc` contains more than one entry.

current_page_name The document name of the current file.

parents, prev and next Information about related chapters in the TOC tree. Each relation is a dictionary with the keys `link` (HREF for the relation) and `title` (title of the related document, as HTML). `parents` is a list of relations, while `prev` and `next` are a single relation.

sourcename The name of the source file under `_sources`.

The special files are located in the root output directory. They are:

SerializingHTMLBuilder.globalcontext_filename A pickled dict with these keys:

project, copyright, release, version The same values as given in the configuration file.

style *html_style*.

last_updated Date of last build.

builder Name of the used builder, in the case of pickles this is always `'pickle'`.

titles A dictionary of all documents' titles, as HTML strings.

SerializingHTMLBuilder.searchindex_filename An index that can be used for searching the documentation. It is a pickled list with these entries:

- A list of indexed docnames.
- A list of document titles, as HTML strings, in the same order as the first list.
- A dict mapping word roots (processed by an English-language stemmer) to a list of integers, which are indices into the first list.

environment.pickle The build environment. This is always a pickle file, independent of the builder and a copy of the environment that was used when the builder was started.

Todo

Document common members.

Unlike the other pickle files this pickle file requires that the `sphinx` package is available on unpickling.

CHAPTER 10

The build configuration file

The *configuration directory* must contain a file named `conf.py`. This file (containing Python code) is called the "build configuration file" and contains all configuration needed to customize Sphinx input and output behavior.

The configuration file is executed as Python code at build time (using `execfile()`, and with the current directory set to its containing directory), and therefore can execute arbitrarily complex code. Sphinx then reads simple names from the file's namespace as its configuration.

Important points to note:

- If not otherwise documented, values must be strings, and their default is the empty string.
- The term "fully-qualified name" refers to a string that names an importable Python object inside a module; for example, the FQN `"sphinx.builders.Builder"` means the `Builder` class in the `sphinx.builders` module.
- Remember that document names use `/` as the path separator and don't contain the file name extension.
- Since `conf.py` is read as a Python file, the usual rules apply for encodings and Unicode support: declare the encoding using an encoding cookie (a comment like `# -*- coding: utf-8 -*-`) and use Unicode string literals when you include non-ASCII characters in configuration values.
- The contents of the config namespace are pickled (so that Sphinx can find out when configuration changes), so it may not contain unpickleable values – delete them from the namespace with `del` if appropriate. Modules are removed automatically, so you don't need to `del` your imports after use.
- There is a special object named `tags` available in the config file. It can be used to query and change the tags (see *Including content based on tags*). Use `tags.has('tag')` to query, `tags.add('tag')` and `tags.remove('tag')` to change. Only tags set via the `-t` command-line option or via `tags.add('tag')` can be queried using `tags.has('tag')`. Note that the current builder tag is not available in `conf.py`, as it is created *after* the builder is initialized.

10.1 General configuration

`extensions`

A list of strings that are module names of Sphinx extensions. These can be extensions coming with Sphinx (named `sphinx.ext.*`) or custom ones.

Note that you can extend `sys.path` within the conf file if your extensions live in another directory – but make sure you use absolute paths. If your extension path is relative to the *configuration directory*, use `os.path.abspath()` like so:

```
import sys, os

sys.path.append(os.path.abspath('sphinxext'))

extensions = ['extname']
```

That way, you can load an extension called `extname` from the subdirectory `sphinxext`.

The configuration file itself can be an extension; for that, you only need to provide a `setup()` function in it.

source_suffix
The file name extension, or list of extensions, of source files. Only files with this suffix will be read as sources. Default is `'.rst'`.

Changed in version 1.3: Can now be a list of extensions.

source_encoding
The encoding of all reST source files. The recommended encoding, and the default value, is `'utf-8-sig'`.

New in version 0.5: Previously, Sphinx accepted only UTF-8 encoded sources.

source_parsers
If given, a dictionary of parser classes for different source suffices. The keys are the suffix, the values can be either a class or a string giving a fully-qualified name of a parser class. Files with a suffix that is not in the dictionary will be parsed with the default reStructuredText parser.

For example:

```
source_parsers = {'.md': 'some.markdown.module.Parser'}
```

New in version 1.3.

master_doc
The document name of the "master" document, that is, the document that contains the root *toctree* directive. Default is `'contents'`.

exclude_patterns
A list of glob-style patterns that should be excluded when looking for source files. [1] They are matched against the source file names relative to the source directory, using slashes as directory separators on all platforms.

Example patterns:

- `'library/xml.rst'` – ignores the `library/xml.rst` file (replaces entry in `unused_docs`)
- `'library/xml'` – ignores the `library/xml` directory (replaces entry in `exclude_trees`)
- `'library/xml*'` – ignores all files and directories starting with `library/xml`
- `'**/.svn'` – ignores all `.svn` directories (replaces entry in `exclude_dirnames`)

exclude_patterns is also consulted when looking for static files in *html_static_path*.

New in version 1.0.

templates_path
A list of paths that contain extra templates (or templates that overwrite builtin/theme-specific templates). Relative paths are taken as relative to the configuration directory.

[1] A note on available globbing syntax: you can use the standard shell constructs `*`, `?`, `[...]` and `[!...]` with the feature that these all don't match slashes. A double star `**` can be used to match any sequence of characters *including* slashes.

Changed in version 1.3: As these files are not meant to be built, they are automatically added to *exclude_patterns*.

template_bridge
A string with the fully-qualified name of a callable (or simply a class) that returns an instance of *TemplateBridge*. This instance is then used to render HTML documents, and possibly the output of other builders (currently the changes builder). (Note that the template bridge must be made theme-aware if HTML themes are to be used.)

rst_epilog
A string of reStructuredText that will be included at the end of every source file that is read. This is the right place to add substitutions that should be available in every file. An example:

```
rst_epilog = """
.. |psf| replace:: Python Software Foundation
"""
```

New in version 0.6.

rst_prolog
A string of reStructuredText that will be included at the beginning of every source file that is read.

New in version 1.0.

primary_domain
The name of the default *domain*. Can also be `None` to disable a default domain. The default is `'py'`. Those objects in other domains (whether the domain name is given explicitly, or selected by a *default-domain* directive) will have the domain name explicitly prepended when named (e.g., when the default domain is C, Python functions will be named "Python function", not just "function").

New in version 1.0.

default_role
The name of a reST role (builtin or Sphinx extension) to use as the default role, that is, for text marked up `'like this'`. This can be set to `'py:obj'` to make `'filter'` a cross-reference to the Python function "filter". The default is `None`, which doesn't reassign the default role.

The default role can always be set within individual documents using the standard reST `default-role` directive.

New in version 0.4.

keep_warnings
If true, keep warnings as "system message" paragraphs in the built documents. Regardless of this setting, warnings are always written to the standard error stream when `sphinx-build` is run.

The default is `False`, the pre-0.5 behavior was to always keep them.

New in version 0.5.

needs_sphinx
If set to a `major.minor` version string like `'1.1'`, Sphinx will compare it with its version and refuse to build if it is too old. Default is no requirement.

New in version 1.0.

needs_extensions
This value can be a dictionary specifying version requirements for extensions in *extensions*, e.g. `needs_extensions = {'sphinxcontrib.something': '1.5'}`. The version strings should be in the form `major.minor`. Requirements do not have to be specified for all extensions, only for those you want to check.

This requires that the extension specifies its version to Sphinx (see *Developing extensions for Sphinx* for how to do that).

New in version 1.3.

nitpicky
If true, Sphinx will warn about *all* references where the target cannot be found. Default is `False`. You can activate this mode temporarily using the `-n` command-line switch.

New in version 1.0.

nitpick_ignore
A list of `(type, target)` tuples (by default empty) that should be ignored when generating warnings in "nitpicky mode". Note that `type` should include the domain name if present. Example entries would be `('py:func', 'int')` or `('envvar', 'LD_LIBRARY_PATH')`.

New in version 1.1.

numfig
If true, figures, tables and code-blocks are automatically numbered if they have a caption. For now, it works only with the HTML builder. Default is `False`.

New in version 1.3.

numfig_format
A dictionary mapping `'figure'`,`'table'` and `'code-block'` to strings that are used for format of figure numbers. Default is to use `'Fig. %s'` for `'figure'`,`'Table %s'` for `'table'` and `'Listing %s'` for `'code-block'`.

New in version 1.3.

numfig_secnum_depth
The scope of figure numbers, that is, the numfig feature numbers figures in which scope. `0` means "whole document". `1` means "in a section". Sphinx numbers like x.1, x.2, x.3... `2` means "in a subsection". Sphinx numbers like x.x.1, x.x.2, x.x.3..., and so on. Default is `1`.

New in version 1.3.

10.2 Project information

project
The documented project's name.

copyright
A copyright statement in the style `'2008, Author Name'`.

version
The major project version, used as the replacement for `|version|`. For example, for the Python documentation, this may be something like `2.6`.

release
The full project version, used as the replacement for `|release|` and e.g. in the HTML templates. For example, for the Python documentation, this may be something like `2.6.0rc1`.

If you don't need the separation provided between *version* and *release*, just set them both to the same value.

today
today_fmt
These values determine how to format the current date, used as the replacement for `|today|`.

- If you set *today* to a non-empty value, it is used.
- Otherwise, the current time is formatted using `time.strftime()` and the format given in *today_fmt*.

The default is no *today* and a *today_fmt* of `'%B %d, %Y'` (or, if translation is enabled with *language*, an equivalent %format for the selected locale).

highlight_language
The default language to highlight source code in. The default is `'python'`. The value should be a valid Pygments lexer name, see *Showing code examples* for more details.

New in version 0.5.

highlight_options
A dictionary of options that modify how the lexer specified by *highlight_language* generates highlighted source code. These are lexer-specific; for the options understood by each, see the Pygments documentation[2].

New in version 1.3.

pygments_style
The style name to use for Pygments highlighting of source code. If not set, either the theme's default style or `'sphinx'` is selected for HTML output.

Changed in version 0.3: If the value is a fully-qualified name of a custom Pygments style class, this is then used as custom style.

add_function_parentheses
A boolean that decides whether parentheses are appended to function and method role text (e.g. the content of `:func:`input``) to signify that the name is callable. Default is `True`.

add_module_names
A boolean that decides whether module names are prepended to all *object* names (for object types where a "module" of some kind is defined), e.g. for *py:function* directives. Default is `True`.

show_authors
A boolean that decides whether *codeauthor* and *sectionauthor* directives produce any output in the built files.

modindex_common_prefix
A list of prefixes that are ignored for sorting the Python module index (e.g., if this is set to `['foo.']`, then `foo.bar` is shown under `B`, not `F`). This can be handy if you document a project that consists of a single package. Works only for the HTML builder currently. Default is `[]`.

New in version 0.6.

trim_footnote_reference_space
Trim spaces before footnote references that are necessary for the reST parser to recognize the footnote, but do not look too nice in the output.

New in version 0.6.

trim_doctest_flags
If true, doctest flags (comments looking like `# doctest: FLAG, ...`) at the ends of lines and `<BLANKLINE>` markers are removed for all code blocks showing interactive Python sessions (i.e. doctests). Default is `True`. See the extension *doctest* for more possibilities of including doctests.

New in version 1.0.

Changed in version 1.1: Now also removes `<BLANKLINE>`.

[2] http://pygments.org/docs/lexers/

10.3 Options for internationalization

These options influence Sphinx's *Native Language Support*. See the documentation on *Internationalization* for details.

language
The code for the language the docs are written in. Any text automatically generated by Sphinx will be in that language. Also, Sphinx will try to substitute individual paragraphs from your documents with the translation sets obtained from `locale_dirs`. In the LaTeX builder, a suitable language will be selected as an option for the *Babel* package. Default is `None`, which means that no translation will be done.

New in version 0.5.

Currently supported languages by Sphinx are:

- `bn` – Bengali
- `ca` – Catalan
- `cs` – Czech
- `da` – Danish
- `de` – German
- `en` – English
- `es` – Spanish
- `et` – Estonian
- `eu` – Basque
- `fa` – Iranian
- `fi` – Finnish
- `fr` – French
- `he` – Hebrew
- `hr` – Croatian
- `hu` – Hungarian
- `id` – Indonesian
- `it` – Italian
- `ja` – Japanese
- `ko` – Korean
- `lt` – Lithuanian
- `lv` – Latvian
- `mk` – Macedonian
- `nb_NO` – Norwegian Bokmal
- `ne` – Nepali
- `nl` – Dutch
- `pl` – Polish

- pt_BR – Brazilian Portuguese
- pt_PT – European Portuguese
- ru – Russian
- si – Sinhala
- sk – Slovak
- sl – Slovenian
- sv – Swedish
- tr – Turkish
- uk_UA – Ukrainian
- vi – Vietnamese
- zh_CN – Simplified Chinese
- zh_TW – Traditional Chinese

locale_dirs
New in version 0.5.

Directories in which to search for additional message catalogs (see *language*), relative to the source directory. The directories on this path are searched by the standard gettext module.

Internal messages are fetched from a text domain of sphinx; so if you add the directory ./locale to this settting, the message catalogs (compiled from .po format using **msgfmt**) must be in ./locale/*language*/LC_MESSAGES/sphinx.mo. The text domain of individual documents depends on *gettext_compact*.

The default is [].

gettext_compact
New in version 1.1.

If true, a document's text domain is its docname if it is a top-level project file and its very base directory otherwise.

By default, the document markup/code.rst ends up in the markup text domain. With this option set to False, it is markup/code.

gettext_uuid
If true, Sphinx generates uuid information for version tracking in message catalogs. It is used for:

- Add uid line for each msgids in .pot files.
- Calculate similarity between new msgids and previously saved old msgids. This calculation takes a long time.

If you want to accelerate the calculation, you can use python-levenshtein 3rd-party package written in C by using **pip install python-levenshtein**.

The default is False.

New in version 1.3.

gettext_location
If true, Sphinx generates location information for messages in message catalogs.

The default is True.

New in version 1.3.

gettext_auto_build
If true, Sphinx builds mo file for each translation catalog files.

The default is `True`.

New in version 1.3.

gettext_additional_targets
To specify names to enable gettext extracting and translation applying for i18n additionally. You can specify below names:

Index index terms

Literal-block literal blocks: `::` and `code-block`.

Doctest-block doctest block

Raw raw content

Image image/figure uri and alt

For example: `gettext_additional_targets = ['literal-block', 'image']`.

The default is `[]`.

New in version 1.3.

10.4 Options for HTML output

These options influence HTML as well as HTML Help output, and other builders that use Sphinx's HTMLWriter class.

html_theme
The "theme" that the HTML output should use. See the section about theming. The default is `'alabaster'`.

New in version 0.6.

html_theme_options
A dictionary of options that influence the look and feel of the selected theme. These are theme-specific. For the options understood by the builtin themes, see *this section*.

New in version 0.6.

html_theme_path
A list of paths that contain custom themes, either as subdirectories or as zip files. Relative paths are taken as relative to the configuration directory.

New in version 0.6.

html_style
The style sheet to use for HTML pages. A file of that name must exist either in Sphinx's `static/` path, or in one of the custom paths given in *`html_static_path`*. Default is the stylesheet given by the selected theme. If you only want to add or override a few things compared to the theme's stylesheet, use CSS `@import` to import the theme's stylesheet.

html_title
The "title" for HTML documentation generated with Sphinx's own templates. This is appended to the `<title>` tag of individual pages, and used in the navigation bar as the "topmost" element. It defaults to `'<project> v<revision> documentation'` (with the values coming from the config values).

html_short_title
A shorter "title" for the HTML docs. This is used in for links in the header and in the HTML Help docs. If not given, it defaults to the value of *html_title*.

New in version 0.4.

html_context
A dictionary of values to pass into the template engine's context for all pages. Single values can also be put in this dictionary using the `-A` command-line option of `sphinx-build`.

New in version 0.5.

html_logo
If given, this must be the name of an image file (path relative to the *configuration directory*) that is the logo of the docs. It is placed at the top of the sidebar; its width should therefore not exceed 200 pixels. Default: `None`.

New in version 0.4.1: The image file will be copied to the `_static` directory of the output HTML, but only if the file does not already exist there.

html_favicon
If given, this must be the name of an image file (path relative to the *configuration directory*) that is the favicon of the docs. Modern browsers use this as the icon for tabs, windows and bookmarks. It should be a Windows-style icon file (`.ico`), which is 16x16 or 32x32 pixels large. Default: `None`.

New in version 0.4: The image file will be copied to the `_static` directory of the output HTML, but only if the file does not already exist there.

html_static_path
A list of paths that contain custom static files (such as style sheets or script files). Relative paths are taken as relative to the configuration directory. They are copied to the output's `_static` directory after the theme's static files, so a file named `default.css` will overwrite the theme's `default.css`.

Changed in version 0.4: The paths in *html_static_path* can now contain subdirectories.

Changed in version 1.0: The entries in *html_static_path* can now be single files.

html_extra_path
A list of paths that contain extra files not directly related to the documentation, such as `robots.txt` or `.htaccess`. Relative paths are taken as relative to the configuration directory. They are copied to the output directory. They will overwrite any existing file of the same name.

As these files are not meant to be built, they are automatically added to *exclude_patterns*.

New in version 1.2.

html_last_updated_fmt
If this is not the empty string, a 'Last updated on:' timestamp is inserted at every page bottom, using the given `strftime()` format. Default is `'%b %d, %Y'` (or a locale-dependent equivalent).

html_use_smartypants
If true, SmartyPants[3] will be used to convert quotes and dashes to typographically correct entities. Default: `True`.

html_add_permalinks
Sphinx will add "permalinks" for each heading and description environment as paragraph signs that become visible when the mouse hovers over them.

This value determines the text for the permalink; it defaults to `"¶"`. Set it to `None` or the empty string to disable permalinks.

[3] http://daringfireball.net/projects/smartypants/

New in version 0.6: Previously, this was always activated.

Changed in version 1.1: This can now be a string to select the actual text of the link. Previously, only boolean values were accepted.

html_sidebars

Custom sidebar templates, must be a dictionary that maps document names to template names.

The keys can contain glob-style patterns [1], in which case all matching documents will get the specified sidebars. (A warning is emitted when a more than one glob-style pattern matches for any document.)

The values can be either lists or single strings.

- If a value is a list, it specifies the complete list of sidebar templates to include. If all or some of the default sidebars are to be included, they must be put into this list as well.

  The default sidebars (for documents that don't match any pattern) are: `['localtoc.html', 'relations.html', 'sourcelink.html', 'searchbox.html']`.

- If a value is a single string, it specifies a custom sidebar to be added between the `'sourcelink.html'` and `'searchbox.html'` entries. This is for compatibility with Sphinx versions before 1.0.

Builtin sidebar templates that can be rendered are:

- **localtoc.html** – a fine-grained table of contents of the current document
- **globaltoc.html** – a coarse-grained table of contents for the whole documentation set, collapsed
- **relations.html** – two links to the previous and next documents
- **sourcelink.html** – a link to the source of the current document, if enabled in *html_show_sourcelink*
- **searchbox.html** – the "quick search" box

Example:

```
html_sidebars = {
   '**': ['globaltoc.html', 'sourcelink.html', 'searchbox.html'],
   'using/windows': ['windowssidebar.html', 'searchbox.html'],
}
```

This will render the custom template `windowssidebar.html` and the quick search box within the sidebar of the given document, and render the default sidebars for all other pages (except that the local TOC is replaced by the global TOC).

New in version 1.0: The ability to use globbing keys and to specify multiple sidebars.

Note that this value only has no effect if the chosen theme does not possess a sidebar, like the builtin **scrolls** and **haiku** themes.

html_additional_pages

Additional templates that should be rendered to HTML pages, must be a dictionary that maps document names to template names.

Example:

```
html_additional_pages = {
    'download': 'customdownload.html',
}
```

This will render the template `customdownload.html` as the page `download.html`.

html_domain_indices
If true, generate domain-specific indices in addition to the general index. For e.g. the Python domain, this is the global module index. Default is `True`.

This value can be a bool or a list of index names that should be generated. To find out the index name for a specific index, look at the HTML file name. For example, the Python module index has the name `'py-modindex'`.

New in version 1.0.

html_use_modindex
If true, add a module index to the HTML documents. Default is `True`.

Deprecated since version 1.0: Use *html_domain_indices*.

html_use_index
If true, add an index to the HTML documents. Default is `True`.

New in version 0.4.

html_split_index
If true, the index is generated twice: once as a single page with all the entries, and once as one page per starting letter. Default is `False`.

New in version 0.4.

html_copy_source
If true, the reST sources are included in the HTML build as `_sources/`*name*. The default is `True`.

> **Warning:** If this config value is set to `False`, the JavaScript search function will only display the titles of matching documents, and no excerpt from the matching contents.

html_show_sourcelink
If true (and *html_copy_source* is true as well), links to the reST sources will be added to the sidebar. The default is `True`.

New in version 0.6.

html_use_opensearch
If nonempty, an OpenSearch[4] description file will be output, and all pages will contain a `<link>` tag referring to it. Since OpenSearch doesn't support relative URLs for its search page location, the value of this option must be the base URL from which these documents are served (without trailing slash), e.g. `"https://docs.python.org"`. The default is `''`.

html_file_suffix
This is the file name suffix for generated HTML files. The default is `".html"`.

New in version 0.4.

html_link_suffix
Suffix for generated links to HTML files. The default is whatever *html_file_suffix* is set to; it can be set differently (e.g. to support different web server setups).

New in version 0.6.

html_translator_class
A string with the fully-qualified name of a HTML Translator class, that is, a subclass of Sphinx's `HTMLTranslator`, that is used to translate document trees to HTML. Default is `None` (use the builtin translator).

See also:

[4] http://opensearch.org

set_translator()

html_show_copyright
If true, "(C) Copyright ..." is shown in the HTML footer. Default is `True`.

New in version 1.0.

html_show_sphinx
If true, "Created using Sphinx" is shown in the HTML footer. Default is `True`.

New in version 0.4.

html_output_encoding
Encoding of HTML output files. Default is `'utf-8'`. Note that this encoding name must both be a valid Python encoding name and a valid HTML `charset` value.

New in version 1.0.

html_compact_lists
If true, list items containing only a single paragraph will not be rendered with a <p> element. This is standard docutils behavior. Default: `True`.

New in version 1.0.

html_secnumber_suffix
Suffix for section numbers. Default: `". "`. Set to `" "` to suppress the final dot on section numbers.

New in version 1.0.

html_search_language
Language to be used for generating the HTML full-text search index. This defaults to the global language selected with *language*. If there is no support for this language, `"en"` is used which selects the English language.

Support is present for these languages:

- `da` – Danish
- `nl` – Dutch
- `en` – English
- `fi` – Finnish
- `fr` – French
- `de` – German
- `hu` – Hungarian
- `it` – Italian
- `ja` – Japanese
- `no` – Norwegian
- `pt` – Portuguese
- `ro` – Romanian
- `ru` – Russian
- `es` – Spanish
- `sv` – Swedish
- `tr` – Turkish

Accelerating build speed

Each language (except Japanese) provides its own stemming algorithm. Sphinx uses a Python implementation by default. You can use a C implementation to accelerate building the index file.

- PorterStemmer[5] (`en`)
- PyStemmer[6] (all languages)

New in version 1.1: With support for `en` and `ja`.

Changed in version 1.3: Added additional languages.

html_search_options
A dictionary with options for the search language support, empty by default. The meaning of these options depends on the language selected.

The English support has no options.

The Japanese support has these options:

- `type` – `'mecab'` or `'default'` (selects either MeCab or TinySegmenter word splitter algorithm)
- `dic_enc` – the encoding for the MeCab algorithm
- `dict` – the dictionary to use for the MeCab algorithm
- `lib` – the library name for finding the MeCab library via ctypes if the Python binding is not installed

New in version 1.1.

html_search_scorer
The name of a JavaScript file (relative to the configuration directory) that implements a search results scorer. If empty, the default will be used.

New in version 1.2.

html_scaled_image_link
If true, images itself links to the original image if it doesn't have 'target' option or scale related options: 'scale', 'width', 'height'. The default is `True`.

New in version 1.3.

htmlhelp_basename
Output file base name for HTML help builder. Default is `'pydoc'`.

10.5 Options for Apple Help output

New in version 1.3.

These options influence the Apple Help output. This builder derives from the HTML builder, so the HTML options also apply where appropriate.

Note: Apple Help output will only work on Mac OS X 10.6 and higher, as it requires the **hiutil** and **codesign** command line tools, neither of which are Open Source.

[5] https://pypi.python.org/pypi/PorterStemmer
[6] https://pypi.python.org/pypi/PyStemmer

You can disable the use of these tools using *applehelp_disable_external_tools*, but the result will not be a valid help book until the indexer is run over the `.lproj` folders within the bundle.

---

applehelp_bundle_name
The basename for the Apple Help Book. Defaults to the *project* name.

applehelp_bundle_id
The bundle ID for the help book bundle.

> **Warning:** You *must* set this value in order to generate Apple Help.

applehelp_dev_region
The development region. Defaults to `'en-us'`, which is Apple's recommended setting.

applehelp_bundle_version
The bundle version (as a string). Defaults to `'1'`.

applehelp_icon
The help bundle icon file, or `None` for no icon. According to Apple's documentation, this should be a 16-by-16 pixel version of the application's icon with a transparent background, saved as a PNG file.

applehelp_kb_product
The product tag for use with *applehelp_kb_url*. Defaults to `'<project>-<release>'`.

applehelp_kb_url
The URL for your knowledgebase server, e.g. `https://example.com/kbsearch.py?p='product'&q='query'`. Help Viewer will replace the values `'product'`, `'query'` and `'lang'` at runtime with the contents of *applehelp_kb_product*, the text entered by the user in the search box and the user's system language respectively.

Defaults to `None` for no remote search.

applehelp_remote_url
The URL for remote content. You can place a copy of your Help Book's `Resources` folder at this location and Help Viewer will attempt to use it to fetch updated content.

e.g. if you set it to `https://example.com/help/Foo/` and Help Viewer wants a copy of `index.html` for an English speaking customer, it will look at `https://example.com/help/Foo/en.lproj/index.html`.

Defaults to `None` for no remote content.

applehelp_index_anchors
If `True`, tell the help indexer to index anchors in the generated HTML. This can be useful for jumping to a particular topic using the `AHLookupAnchor` function or the `openHelpAnchor:inBook:` method in your code. It also allows you to use `help:anchor` URLs; see the Apple documentation for more information on this topic.

applehelp_min_term_length
Controls the minimum term length for the help indexer. Defaults to `None`, which means the default will be used.

applehelp_stopwords
Either a language specification (to use the built-in stopwords), or the path to a stopwords plist, or `None` if you do not want to use stopwords. The default stopwords plist can be found at `/usr/share/hiutil/Stopwords.plist` and contains, at time of writing, stopwords for the following languages:

Language	Code
English	en
German	de
Spanish	es
French	fr
Swedish	sv
Hungarian	hu
Italian	it

Defaults to *language*, or if that is not set, to en.

applehelp_locale
Specifies the locale to generate help for. This is used to determine the name of the .lproj folder inside the Help Book's Resources, and is passed to the help indexer.

Defaults to *language*, or if that is not set, to en.

applehelp_title
Specifies the help book title. Defaults to '*<project>* Help'.

applehelp_codesign_identity
Specifies the identity to use for code signing, or None if code signing is not to be performed.

Defaults to the value of the environment variable CODE_SIGN_IDENTITY, which is set by Xcode for script build phases, or None if that variable is not set.

applehelp_codesign_flags
A *list* of additional arguments to pass to **codesign** when signing the help book.

Defaults to a list based on the value of the environment variable OTHER_CODE_SIGN_FLAGS, which is set by Xcode for script build phases, or the empty list if that variable is not set.

applehelp_indexer_path
The path to the **hiutil** program. Defaults to '/usr/bin/hiutil'.

applehelp_codesign_path
The path to the **codesign** program. Defaults to '/usr/bin/codesign'.

applehelp_disable_external_tools
If True, the builder will not run the indexer or the code signing tool, no matter what other settings are specified.

This is mainly useful for testing, or where you want to run the Sphinx build on a non-Mac OS X platform and then complete the final steps on OS X for some reason.

Defaults to False.

10.6 Options for epub output

These options influence the epub output. As this builder derives from the HTML builder, the HTML options also apply where appropriate. The actual values for some of the options is not really important, they just have to be entered into the Dublin Core metadata[7].

epub_basename
The basename for the epub file. It defaults to the *project* name.

[7] http://dublincore.org/

epub_theme
The HTML theme for the epub output. Since the default themes are not optimized for small screen space, using the same theme for HTML and epub output is usually not wise. This defaults to `'epub'`, a theme designed to save visual space.

epub_theme_options
A dictionary of options that influence the look and feel of the selected theme. These are theme-specific. For the options understood by the builtin themes, see *this section*.

New in version 1.2.

epub_title
The title of the document. It defaults to the *html_title* option but can be set independently for epub creation.

epub_author
The author of the document. This is put in the Dublin Core metadata. The default value is `'unknown'`.

epub_language
The language of the document. This is put in the Dublin Core metadata. The default is the *language* option or `'en'` if unset.

epub_publisher
The publisher of the document. This is put in the Dublin Core metadata. You may use any sensible string, e.g. the project homepage. The default value is `'unknown'`.

epub_copyright
The copyright of the document. It defaults to the *copyright* option but can be set independently for epub creation.

epub_identifier
An identifier for the document. This is put in the Dublin Core metadata. For published documents this is the ISBN number, but you can also use an alternative scheme, e.g. the project homepage. The default value is `'unknown'`.

epub_scheme
The publication scheme for the *epub_identifier*. This is put in the Dublin Core metadata. For published books the scheme is `'ISBN'`. If you use the project homepage, `'URL'` seems reasonable. The default value is `'unknown'`.

epub_uid
A unique identifier for the document. This is put in the Dublin Core metadata. You may use a random string. The default value is `'unknown'`.

epub_cover
The cover page information. This is a tuple containing the filenames of the cover image and the html template. The rendered html cover page is inserted as the first item in the spine in `content.opf`. If the template filename is empty, no html cover page is created. No cover at all is created if the tuple is empty. Examples:

```
epub_cover = ('_static/cover.png', 'epub-cover.html')
epub_cover = ('_static/cover.png', '')
epub_cover = ()
```

The default value is `()`.

New in version 1.1.

epub_guide
Meta data for the guide element of `content.opf`. This is a sequence of tuples containing the

type, the *uri* and the *title* of the optional guide information. See the OPF documentation at http://idpf.org/epub for details. If possible, default entries for the *cover* and *toc* types are automatically inserted. However, the types can be explicitly overwritten if the default entries are not appropriate. Example:

```
epub_guide = (('cover', 'cover.html', u'Cover Page'),)
```

The default value is `()`.

epub_pre_files
Additional files that should be inserted before the text generated by Sphinx. It is a list of tuples containing the file name and the title. If the title is empty, no entry is added to `toc.ncx`. Example:

```
epub_pre_files = [
    ('index.html', 'Welcome'),
]
```

The default value is `[]`.

epub_post_files
Additional files that should be inserted after the text generated by Sphinx. It is a list of tuples containing the file name and the title. This option can be used to add an appendix. If the title is empty, no entry is added to `toc.ncx`. The default value is `[]`.

epub_exclude_files
A list of files that are generated/copied in the build directory but should not be included in the epub file. The default value is `[]`.

epub_tocdepth
The depth of the table of contents in the file `toc.ncx`. It should be an integer greater than zero. The default value is 3. Note: A deeply nested table of contents may be difficult to navigate.

epub_tocdup
This flag determines if a toc entry is inserted again at the beginning of its nested toc listing. This allows easier navigation to the top of a chapter, but can be confusing because it mixes entries of different depth in one list. The default value is `True`.

epub_tocscope
This setting control the scope of the epub table of contents. The setting can have the following values:

- `'default'` – include all toc entries that are not hidden (default)
- `'includehidden'` – include all toc entries

New in version 1.2.

epub_fix_images
This flag determines if sphinx should try to fix image formats that are not supported by some epub readers. At the moment palette images with a small color table are upgraded. You need the Python Image Library (Pillow the successor of the PIL) installed to use this option. The default value is `False` because the automatic conversion may lose information.

New in version 1.2.

epub_max_image_width
This option specifies the maximum width of images. If it is set to a value greater than zero, images with a width larger than the given value are scaled accordingly. If it is zero, no scaling is performed. The default value is `0`. You need the Python Image Library (Pillow) installed to use this option.

New in version 1.2.

epub_show_urls
Control whether to display URL addresses. This is very useful for readers that have no other means to display the linked URL. The settings can have the following values:

- `'inline'` – display URLs inline in parentheses (default)
- `'footnote'` – display URLs in footnotes
- `'no'` – do not display URLs

The display of inline URLs can be customized by adding CSS rules for the class `link-target`.

New in version 1.2.

epub_use_index
If true, add an index to the epub document. It defaults to the *html_use_index* option but can be set independently for epub creation.

New in version 1.2.

10.7 Options for LaTeX output

These options influence LaTeX output.

latex_documents
This value determines how to group the document tree into LaTeX source files. It must be a list of tuples `(startdocname, targetname, title, author, documentclass, toctree_only)`, where the items are:

- *startdocname*: document name that is the "root" of the LaTeX file. All documents referenced by it in TOC trees will be included in the LaTeX file too. (If you want only one LaTeX file, use your *master_doc* here.)
- *targetname*: file name of the LaTeX file in the output directory.
- *title*: LaTeX document title. Can be empty to use the title of the *startdoc*. This is inserted as LaTeX markup, so special characters like a backslash or ampersand must be represented by the proper LaTeX commands if they are to be inserted literally.
- *author*: Author for the LaTeX document. The same LaTeX markup caveat as for *title* applies. Use `\and` to separate multiple authors, as in: `'John \and Sarah'`.
- *documentclass*: Normally, one of `'manual'` or `'howto'` (provided by Sphinx). Other document classes can be given, but they must include the "sphinx" package in order to define Sphinx's custom LaTeX commands. "howto" documents will not get appendices. Also, howtos will have a simpler title page.
- *toctree_only*: Must be `True` or `False`. If true, the *startdoc* document itself is not included in the output, only the documents referenced by it via TOC trees. With this option, you can put extra stuff in the master document that shows up in the HTML, but not the LaTeX output.

New in version 1.2: In the past including your own document class required you to prepend the document class name with the string "sphinx". This is not necessary anymore.

New in version 0.3: The 6th item `toctree_only`. Tuples with 5 items are still accepted.

latex_logo
If given, this must be the name of an image file (relative to the configuration directory) that is the logo of the docs. It is placed at the top of the title page. Default: `None`.

latex_use_parts
If true, the topmost sectioning unit is parts, else it is chapters. Default: `False`.

New in version 0.3.

latex_appendices
A list of document names to append as an appendix to all manuals.

latex_domain_indices
If true, generate domain-specific indices in addition to the general index. For e.g. the Python domain, this is the global module index. Default is `True`.

This value can be a bool or a list of index names that should be generated, like for *html_domain_indices*.

New in version 1.0.

latex_use_modindex
If true, add a module index to LaTeX documents. Default is `True`.

Deprecated since version 1.0: Use *latex_domain_indices*.

latex_show_pagerefs
If true, add page references after internal references. This is very useful for printed copies of the manual. Default is `False`.

New in version 1.0.

latex_show_urls
Control whether to display URL addresses. This is very useful for printed copies of the manual. The setting can have the following values:

- `'no'` – do not display URLs (default)
- `'footnote'` – display URLs in footnotes
- `'inline'` – display URLs inline in parentheses

New in version 1.0.

Changed in version 1.1: This value is now a string; previously it was a boolean value, and a true value selected the `'inline'` display. For backwards compatibility, `True` is still accepted.

latex_elements
New in version 0.5.

A dictionary that contains LaTeX snippets that override those Sphinx usually puts into the generated `.tex` files.

Keep in mind that backslashes must be doubled in Python string literals to avoid interpretation as escape sequences.

- Keys that you may want to override include:

'papersize' Paper size option of the document class (`'a4paper'` or `'letterpaper'`), default `'letterpaper'`.

'pointsize' Point size option of the document class (`'10pt'`, `'11pt'` or `'12pt'`), default `'10pt'`.

'babel' "babel" package inclusion, default `'\\usepackage{babel}'`.

'fontpkg' Font package inclusion, default `'\\usepackage{times}'` (which uses Times and Helvetica). You can set this to `''` to use the Computer Modern fonts.

Changed in version 1.2: Defaults to `''` when the *language* uses the Cyrillic script.

'fncychap' Inclusion of the "fncychap" package (which makes fancy chapter titles), default `'\\usepackage[Bjarne]{fncychap}'` for English documentation, `'\\usepackage[Sonny]{fncychap}'` for internationalized docs (because the "Bjarne" style uses numbers spelled out in English). Other "fncychap" styles you can try include "Lenny", "Glenn", "Conny" and "Rejne". You can also set this to `''` to disable fncychap.

'preamble' Additional preamble content, default empty.

'figure_align' Latex figure float alignment, default 'htbp' (here, top, bottom, page). Whenever an image doesn't fit into the current page, it will be 'floated' into the next page but may be preceded by any other text. If you don't like this behavior, use 'H' which will disable floating and position figures strictly in the order they appear in the source.

'footer' Additional footer content (before the indices), default empty.

- Keys that don't need be overridden unless in special cases are:

'inputenc' "inputenc" package inclusion, default `'\\usepackage[utf8]{inputenc}'`.

'cmappkg' "cmap" package inclusion, default `'\\usepackage{cmap}'`.

New in version 1.2.

'fontenc' "fontenc" package inclusion, default `'\\usepackage[T1]{fontenc}'`.

'maketitle' "maketitle" call, default `'\\maketitle'`. Override if you want to generate a differently-styled title page.

'releasename' value that prefixes `'release'` element on title page, default `'Release'`.

'tableofcontents' "tableofcontents" call, default `'\\tableofcontents'`. Override if you want to generate a different table of contents or put content between the title page and the TOC.

'transition' Commands used to display transitions, default `'\n\n\\bigskip\\hrule{}\\bigskip\n\n'`. Override if you want to display transitions differently.

New in version 1.2.

'printindex' "printindex" call, the last thing in the file, default `'\\printindex'`. Override if you want to generate the index differently or append some content after the index.

- Keys that are set by other options and therefore should not be overridden are:

`'docclass' 'classoptions' 'title' 'date' 'release' 'author' 'logo' 'makeindex' 'shorthandoff'`

latex_docclass

A dictionary mapping `'howto'` and `'manual'` to names of real document classes that will be used as the base for the two Sphinx classes. Default is to use `'article'` for `'howto'` and `'report'` for `'manual'`.

New in version 1.0.

latex_additional_files

A list of file names, relative to the configuration directory, to copy to the build directory when building LaTeX output. This is useful to copy files that Sphinx doesn't copy automatically, e.g. if they are referenced in custom LaTeX added in `latex_elements`. Image files that are referenced in source files (e.g. via `.. image::`) are copied automatically.

You have to make sure yourself that the filenames don't collide with those of any automatically copied files.

New in version 0.6.

Changed in version 1.2: This overrides the files which is provided from Sphinx such as sphinx.sty.

latex_preamble
Additional LaTeX markup for the preamble.

Deprecated since version 0.5: Use the `'preamble'` key in the *latex_elements* value.

latex_paper_size
The output paper size (`'letter'` or `'a4'`). Default is `'letter'`.

Deprecated since version 0.5: Use the `'papersize'` key in the *latex_elements* value.

latex_font_size
The font size ('10pt', '11pt' or '12pt'). Default is `'10pt'`.

Deprecated since version 0.5: Use the `'pointsize'` key in the *latex_elements* value.

10.8 Options for text output

These options influence text output.

text_newlines
Determines which end-of-line character(s) are used in text output.

- `'unix'`: use Unix-style line endings (`\n`)
- `'windows'`: use Windows-style line endings (`\r\n`)
- `'native'`: use the line ending style of the platform the documentation is built on

Default: `'unix'`.

New in version 1.1.

text_sectionchars
A string of 7 characters that should be used for underlining sections. The first character is used for first-level headings, the second for second-level headings and so on.

The default is `'*=-~"+`'`.

New in version 1.1.

10.9 Options for manual page output

These options influence manual page output.

man_pages
This value determines how to group the document tree into manual pages. It must be a list of tuples `(startdocname, name, description, authors, section)`, where the items are:

- *startdocname*: document name that is the "root" of the manual page. All documents referenced by it in TOC trees will be included in the manual file too. (If you want one master manual page, use your *master_doc* here.)
- *name*: name of the manual page. This should be a short string without spaces or special characters. It is used to determine the file name as well as the name of the manual page (in the NAME section).

- *description*: description of the manual page. This is used in the NAME section.
- *authors*: A list of strings with authors, or a single string. Can be an empty string or list if you do not want to automatically generate an AUTHORS section in the manual page.
- *section*: The manual page section. Used for the output file name as well as in the manual page header.

New in version 1.0.

man_show_urls
If true, add URL addresses after links. Default is `False`.

New in version 1.1.

10.10 Options for Texinfo output

These options influence Texinfo output.

texinfo_documents
This value determines how to group the document tree into Texinfo source files. It must be a list of tuples `(startdocname, targetname, title, author, dir_entry, description, category, toctree_only)`, where the items are:

- *startdocname*: document name that is the "root" of the Texinfo file. All documents referenced by it in TOC trees will be included in the Texinfo file too. (If you want only one Texinfo file, use your *master_doc* here.)
- *targetname*: file name (no extension) of the Texinfo file in the output directory.
- *title*: Texinfo document title. Can be empty to use the title of the *startdoc*. Inserted as Texinfo markup, so special characters like @ and {} will need to be escaped to be inserted literally.
- *author*: Author for the Texinfo document. Inserted as Texinfo markup. Use `@*` to separate multiple authors, as in: `'John@*Sarah'`.
- *dir_entry*: The name that will appear in the top-level `DIR` menu file.
- *description*: Descriptive text to appear in the top-level `DIR` menu file.
- *category*: Specifies the section which this entry will appear in the top-level `DIR` menu file.
- *toctree_only*: Must be `True` or `False`. If true, the *startdoc* document itself is not included in the output, only the documents referenced by it via TOC trees. With this option, you can put extra stuff in the master document that shows up in the HTML, but not the Texinfo output.

New in version 1.1.

texinfo_appendices
A list of document names to append as an appendix to all manuals.

New in version 1.1.

texinfo_domain_indices
If true, generate domain-specific indices in addition to the general index. For e.g. the Python domain, this is the global module index. Default is `True`.

This value can be a bool or a list of index names that should be generated, like for *html_domain_indices*.

New in version 1.1.

texinfo_show_urls
Control how to display URL addresses.

- `'footnote'` – display URLs in footnotes (default)
- `'no'` – do not display URLs
- `'inline'` – display URLs inline in parentheses

New in version 1.1.

texinfo_no_detailmenu
If true, do not generate a `@detailmenu` in the "Top" node's menu containing entries for each sub-node in the document. Default is `False`.

New in version 1.2.

texinfo_elements
A dictionary that contains Texinfo snippets that override those Sphinx usually puts into the generated `.texi` files.

- Keys that you may want to override include:

  'paragraphindent' Number of spaces to indent the first line of each paragraph, default `2`. Specify `0` for no indentation.

  'exampleindent' Number of spaces to indent the lines for examples or literal blocks, default `4`. Specify `0` for no indentation.

  'preamble' Texinfo markup inserted near the beginning of the file.

  'copying' Texinfo markup inserted within the `@copying` block and displayed after the title. The default value consists of a simple title page identifying the project.

- Keys that are set by other options and therefore should not be overridden are:

  `'author' 'body' 'date' 'direntry' 'filename' 'project' 'release' 'title'`

New in version 1.1.

10.11 Options for the linkcheck builder

linkcheck_ignore
A list of regular expressions that match URIs that should not be checked when doing a `linkcheck` build. Example:

```
linkcheck_ignore = [r'http://localhost:\d+/']
```

New in version 1.1.

linkcheck_timeout
A timeout value, in seconds, for the linkcheck builder. **Only works in Python 2.6 and higher.** The default is to use Python's global socket timeout.

New in version 1.1.

linkcheck_workers
The number of worker threads to use when checking links. Default is 5 threads.

New in version 1.1.

linkcheck_anchors
If true, check the validity of `#anchor`s in links. Since this requires downloading the whole document, it's considerably slower when enabled. Default is `True`.

New in version 1.2.

10.12 Options for the XML builder

xml_pretty
If true, pretty-print the XML. Default is `True`.

New in version 1.2.

CHAPTER 11

Internationalization

New in version 1.1.

Complementary to translations provided for Sphinx-generated messages such as navigation bars, Sphinx provides mechanisms facilitating *document* translations in itself. See the *Options for internationalization* for details on configuration.

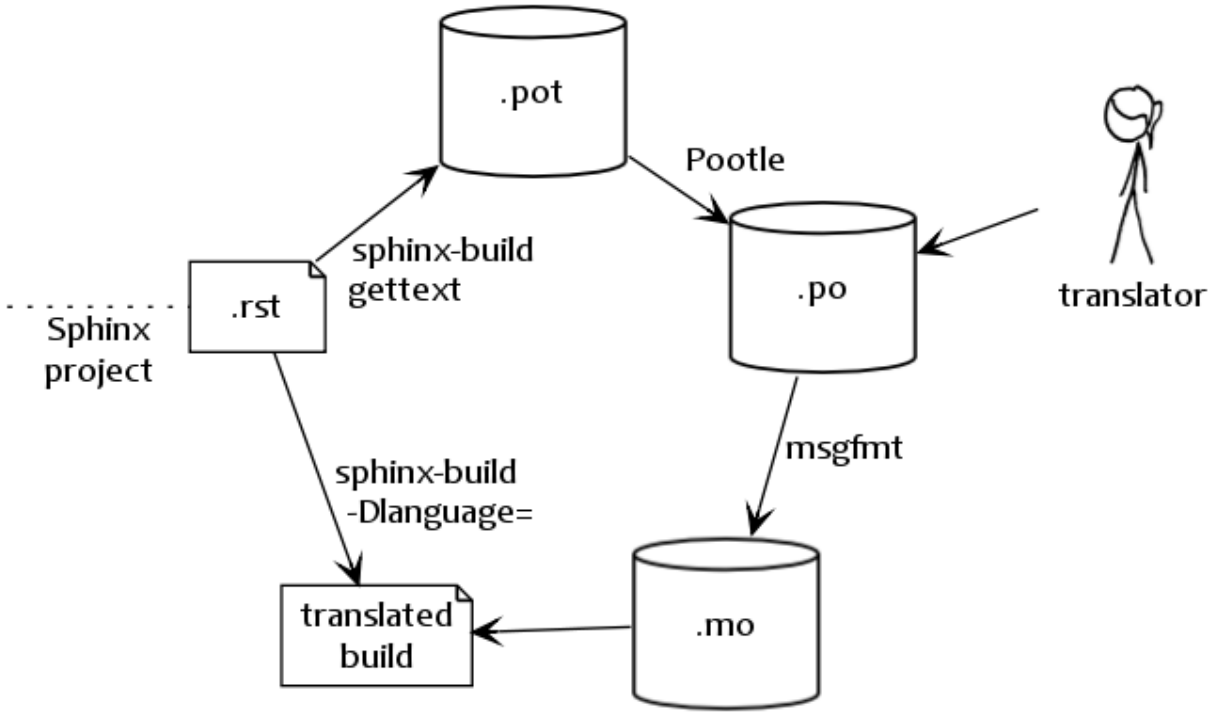


Fig. 11.1: Workflow visualization of translations in Sphinx. (The stick-figure is taken from an XKCD comic (http://xkcd.com/779/).)

- *Sphinx internationalization details*
- *Translating with sphinx-intl*
  - *Quick guide*
  - *Translating*
  - *Update your po files by new pot files*
- *Using Transifex service for team translation*
- *Contributing to Sphinx reference translation*

11.1 Sphinx internationalization details

gettext [1] is an established standard for internationalization and localization. It naively maps messages in a program to a translated string. Sphinx uses these facilities to translate whole documents.

Initially project maintainers have to collect all translatable strings (also referred to as *messages*) to make them known to translators. Sphinx extracts these through invocation of `sphinx-build -b gettext`.

Every single element in the doctree will end up in a single message which results in lists being equally split into different chunks while large paragraphs will remain as coarsely-grained as they were in the original document. This grants seamless document updates while still providing a little bit of context for translators in free-text passages. It is the maintainer's task to split up paragraphs which are too large as there is no sane automated way to do that.

After Sphinx successfully ran the *MessageCatalogBuilder* you will find a collection of `.pot` files in your output directory. These are **catalog templates** and contain messages in your original language *only*.

They can be delivered to translators which will transform them to `.po` files — so called **message catalogs** — containing a mapping from the original messages to foreign-language strings.

Gettext compiles them into a binary format known as **binary catalogs** through **`msgfmt`** for efficiency reasons. If you make these files discoverable with *locale_dirs* for your *language*, Sphinx will pick them up automatically.

An example: you have a document `usage.rst` in your Sphinx project. The gettext builder will put its messages into `usage.pot`. Imagine you have Spanish translations [2] on your hands in `usage.po` — for your builds to be translated you need to follow these instructions:

- Compile your message catalog to a locale directory, say `locale`, so it ends up in `./locale/es/LC_MESSAGES/usage.mo` in your source directory (where `es` is the language code for Spanish.)

```
msgfmt "usage.po" -o "locale/es/LC_MESSAGES/usage.mo"
```

- Set *locale_dirs* to `["locale/"]`.
- Set *language* to `es` (also possible via `-D`).
- Run your desired build.

[1] See the GNU gettext utilities (http://www.gnu.org/software/gettext/manual/gettext.html#Introduction) for details on that software suite.

[2] Because nobody expects the Spanish Inquisition!

11.2 Translating with sphinx-intl

11.2.1 Quick guide

sphinx-intl[3] is a useful tool to work with Sphinx translation flow. This section describe a easy way to translate with sphinx-intl.

1. Install sphinx-intl[4] by **pip install sphinx-intl** or **easy_install sphinx-intl**.
2. Add configurations to your *conf.py*:

```
locale_dirs = ['locale/']   # path is example but recommended.
gettext_compact = False     # optional.
```

   This case-study assumes that *locale_dirs* is set to 'locale/' and *gettext_compact* is set to *False* (the Sphinx document is already configured as such).

3. Extract document's translatable messages into pot files:

```
$ make gettext
```

   As a result, many pot files are generated under `_build/locale` directory.

4. Setup/Update your *locale_dir*:

```
$ sphinx-intl update -p _build/locale -l de -l ja
```

   Done. You got these directories that contain po files:

   - *./locale/de/LC_MESSAGES/*
   - *./locale/ja/LC_MESSAGES/*

5. Translate your po files under *./locale/<lang>/LC_MESSAGES/*.
6. Build mo files and make translated document.

   You need a *language* parameter in `conf.py` or you may also specify the parameter on the command line:

```
$ sphinx-intl build
$ make -e SPHINXOPTS="-D language='de'" html
```

Congratulations! You got the translated documentation in the `_build/html` directory.

11.2.2 Translating

Translate po file under `./locale/de/LC_MESSAGES` directory. The case of builders.po file for sphinx document:

```
# a5600c3d2e3d48fc8c261ea0284db79b
#: ../../builders.rst:4
msgid "Available builders"
msgstr "<FILL HERE BY TARGET LANGUAGE>"
```

Another case, msgid is multi-line text and contains reStructuredText syntax:

[3] https://pypi.python.org/pypi/sphinx-intl
[4] https://pypi.python.org/pypi/sphinx-intl

```
# 302558364e1d41c69b3277277e34b184
#: ../../builders.rst:9
msgid ""
"These are the built-in Sphinx builders. More builders can be added by "
":ref:`extensions <extensions>`."
msgstr ""
"FILL HERE BY TARGET LANGUAGE FILL HERE BY TARGET LANGUAGE FILL HERE "
"BY TARGET LANGUAGE :ref:`EXTENSIONS <extensions>` FILL HERE."
```

Please be careful not to break reST notation. Most po-editors will help you with that.

11.2.3 Update your po files by new pot files

If a document is updated, it is necessary to generate updated pot files and to apply differences to translated po files. In order to apply the updating difference of a pot file to po file, use the **`sphinx-intl update`** command.

```
$ sphinx-intl update -p _build/locale
```

11.3 Using Transifex service for team translation

Transifex[5] is one of several services that allow collaborative translation via a web interface. It has a nifty Python-based command line client that makes it easy to fetch and push translations.

1. Install transifex-client[6]

   You need **`tx`** command to upload resources (pot files).

   ```
   $ pip install transifex-client
   ```

   See also:

   Transifex Client v0.8 — Transifex documentation[7]

2. Create your transifex[8] account and create new project for your document

   Currently, transifex does not allow for a translation project to have more than one version of the document, so you'd better include a version number in your project name.

   For example:

   Project ID `sphinx-document-test_1_0`

   Project URL `https://www.transifex.com/projects/p/sphinx-document-test_1_0/`

3. Create config files for tx command

   This process will create `.tx/config` in the current directory, as well as a `~/.transifexrc` file that includes auth information.

   ```
   $ tx init
   Creating .tx folder...
   Transifex instance [https://www.transifex.com]:
   ...
   ```

[5] https://www.transifex.com/
[6] https://pypi.python.org/pypi/transifex-client
[7] http://help.transifex.com/features/client/index.html
[8] https://www.transifex.com/

```
Please enter your transifex username: <transifex-username>
Password: <transifex-password>
...
Done.
```

4. Upload pot files to transifex service

   Register pot files to `.tx/config` file:

```
$ cd /your/document/root
$ sphinx-intl update-txconfig-resources --pot-dir _build/locale \
  --transifex-project-name sphinx-document-test_1_0
```

   and upload pot files:

```
$ tx push -s
Pushing translations for resource sphinx-document-test_1_0.builders:
Pushing source file (locale/pot/builders.pot)
Resource does not exist.  Creating...
...
Done.
```

5. Forward the translation on transifex
6. Pull translated po files and make translated html

   Get translated catalogs and build mo files (ex. for 'de'):

```
$ cd /your/document/root
$ tx pull -l de
Pulling translations for resource sphinx-document-test_1_0.builders (...)
 -> de: locale/de/LC_MESSAGES/builders.po
...
Done.
```

   Build po files into mo and make html:

```
$ sphinx-intl build
$ make -e SPHINXOPTS="-D language='de'" html
```

That's all!

Tip: Translating locally and on Transifex

If you want to push all language's po files, you can be done by using **`tx push -t`** command. Watch out! This operation overwrites translations in transifex.

In other words, if you have updated each in the service and local po files, it would take much time and effort to integrate them.

11.4 Contributing to Sphinx reference translation

The recommended way for new contributors to translate Sphinx reference is to join the translation team on Transifex.

There is sphinx translation page[9] for Sphinx-1.3 documentation.

[9] https://www.transifex.com/projects/p/sphinx-doc-1_3/

1. Login to transifex[10] service.
2. Go to sphinx translation page[11].
3. Click `Request language` and fill form.
4. Wait acceptance by transifex sphinx translation maintainers.
5. (after acceptance) translate on transifex.

[10] https://www.transifex.com/
[11] https://www.transifex.com/projects/p/sphinx-doc-1_3/

CHAPTER 12

HTML theming support

New in version 0.6.

Sphinx supports changing the appearance of its HTML output via *themes*. A theme is a collection of HTML templates, stylesheet(s) and other static files. Additionally, it has a configuration file which specifies from which theme to inherit, which highlighting style to use, and what options exist for customizing the theme's look and feel.

Themes are meant to be project-unaware, so they can be used for different projects without change.

12.1 Using a theme

Using an existing theme is easy. If the theme is builtin to Sphinx, you only need to set the *html_theme* config value. With the *html_theme_options* config value you can set theme-specific options that change the look and feel. For example, you could have the following in your `conf.py`:

```
html_theme = "classic"
html_theme_options = {
    "rightsidebar": "true",
    "relbarbgcolor": "black"
}
```

That would give you the classic theme, but with a sidebar on the right side and a black background for the relation bar (the bar with the navigation links at the page's top and bottom).

If the theme does not come with Sphinx, it can be in two static forms: either a directory (containing `theme.conf` and other needed files), or a zip file with the same contents. Either of them must be put where Sphinx can find it; for this there is the config value *html_theme_path*. It gives a list of directories, relative to the directory containing `conf.py`, that can contain theme directories or zip files. For example, if you have a theme in the file `blue.zip`, you can put it right in the directory containing `conf.py` and use this configuration:

```
html_theme = "blue"
html_theme_path = ["."]
```

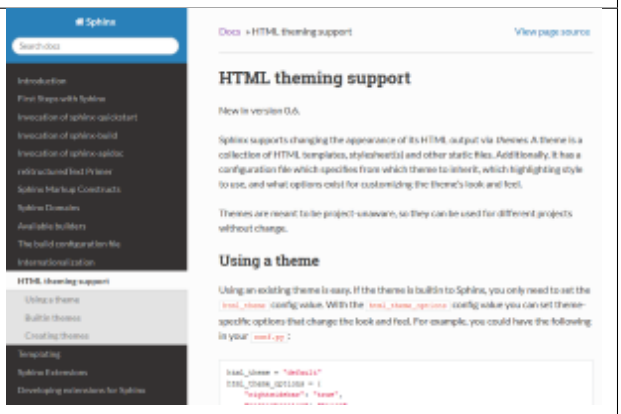
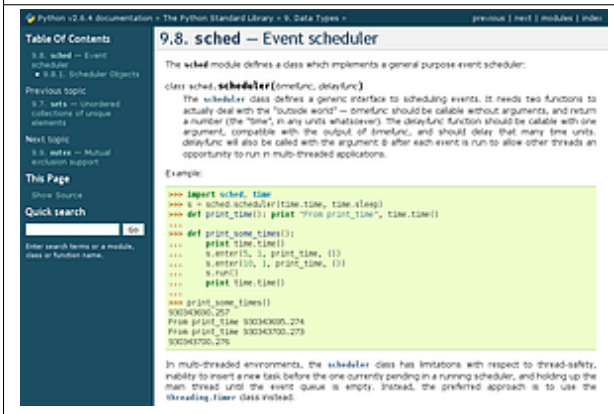
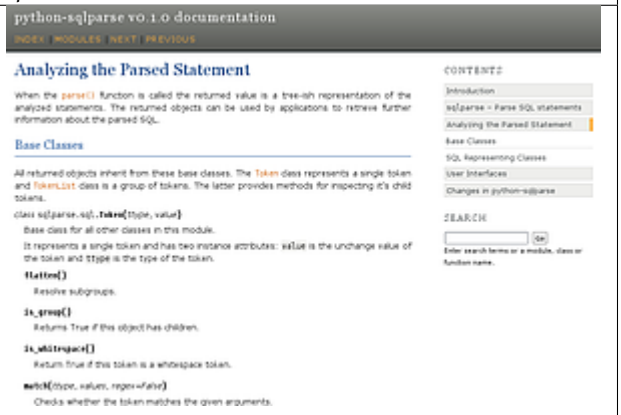
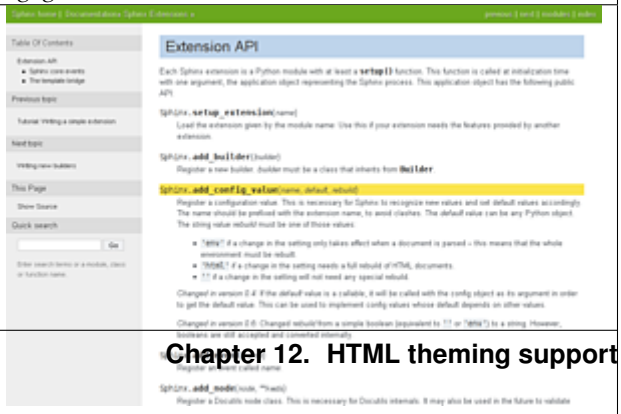
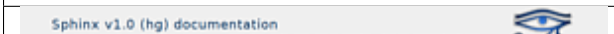
The third form provides your theme path dynamically to Sphinx if the `setuptools` package is installed. You can provide an entry point section called `sphinx_themes` in your setup.py file and write a `get_path` function that has to return the directory with themes in it:

```
// in your 'setup.py'

setup(
```

```
    ...
    entry_points = {
        'sphinx_themes': [
            'path = your_package:get_path',
        ]
    },
    ...
)

// in 'your_package.py'

from os import path
package_dir = path.abspath(path.dirname(__file__))
template_path = path.join(package_dir, 'themes')

def get_path():
    return template_path
```

New in version 1.2: 'sphinx_themes' entry_points feature.

12.2 Builtin themes

Theme overview	
	
alabaster	*sphinx_rtd_theme*
	
classic	*sphinxdoc*
	
scrolls	*agogo*
	
traditional	*nature*
	

Sphinx comes with a selection of themes to choose from.

These themes are:

- **basic** – This is a basically unstyled layout used as the base for the other themes, and usable as the base for custom themes as well. The HTML contains all important elements like sidebar and relation bar. There are these options (which are inherited by the other themes):
  - **nosidebar** (true or false): Don't include the sidebar. Defaults to `False`.
  - **sidebarwidth** (an integer): Width of the sidebar in pixels. (Do not include `px` in the value.) Defaults to 230 pixels.
- **alabaster** – Alabaster theme[1] is a modified "Kr" Sphinx theme from @kennethreitz (especially as used in his Requests project), which was itself originally based on @mitsuhiko's theme used for Flask & related projects. You can get options information at Alabaster theme[2] page.
- **sphinx_rtd_theme** – Read the Docs Sphinx Theme[3]. This is a mobile-friendly sphinx theme that was made for readthedocs.org. View a working demo over on readthedocs.org. You can get options information at Read the Docs Sphinx Theme[4] page.
- **classic** – This is the classic theme, which looks like the Python documentation[5]. It can be customized via these options:
  - **rightsidebar** (true or false): Put the sidebar on the right side. Defaults to `False`.
  - **stickysidebar** (true or false): Make the sidebar "fixed" so that it doesn't scroll out of view for long body content. This may not work well with all browsers. Defaults to `False`.
  - **collapsiblesidebar** (true or false): Add an *experimental* JavaScript snippet that makes the sidebar collapsible via a button on its side. *Doesn't work with "stickysidebar".* Defaults to `False`.
  - **externalrefs** (true or false): Display external links differently from internal links. Defaults to `False`.

  There are also various color and font options that can change the color scheme without having to write a custom stylesheet:
  - **footerbgcolor** (CSS color): Background color for the footer line.
  - **footertextcolor** (CSS color): Text color for the footer line.
  - **sidebarbgcolor** (CSS color): Background color for the sidebar.
  - **sidebarbtncolor** (CSS color): Background color for the sidebar collapse button (used when *collapsiblesidebar* is `True`).
  - **sidebartextcolor** (CSS color): Text color for the sidebar.
  - **sidebarlinkcolor** (CSS color): Link color for the sidebar.
  - **relbarbgcolor** (CSS color): Background color for the relation bar.
  - **relbartextcolor** (CSS color): Text color for the relation bar.
  - **relbarlinkcolor** (CSS color): Link color for the relation bar.
  - **bgcolor** (CSS color): Body background color.
  - **textcolor** (CSS color): Body text color.

[1] https://pypi.python.org/pypi/alabaster
[2] https://pypi.python.org/pypi/alabaster
[3] https://pypi.python.org/pypi/sphinx_rtd_theme
[4] https://pypi.python.org/pypi/sphinx_rtd_theme
[5] https://docs.python.org/

  - **linkcolor** (CSS color): Body link color.
  - **visitedlinkcolor** (CSS color): Body color for visited links.
  - **headbgcolor** (CSS color): Background color for headings.
  - **headtextcolor** (CSS color): Text color for headings.
  - **headlinkcolor** (CSS color): Link color for headings.
  - **codebgcolor** (CSS color): Background color for code blocks.
  - **codetextcolor** (CSS color): Default text color for code blocks, if not set differently by the highlighting style.
  - **bodyfont** (CSS font-family): Font for normal text.
  - **headfont** (CSS font-family): Font for headings.

- **sphinxdoc** – The theme used for this documentation. It features a sidebar on the right side. There are currently no options beyond *nosidebar* and *sidebarwidth*.
- **scrolls** – A more lightweight theme, based on the Jinja documentation[6]. The following color options are available:
  - **headerbordercolor**
  - **subheadlinecolor**
  - **linkcolor**
  - **visitedlinkcolor**
  - **admonitioncolor**
- **agogo** – A theme created by Andi Albrecht. The following options are supported:
  - **bodyfont** (CSS font family): Font for normal text.
  - **headerfont** (CSS font family): Font for headings.
  - **pagewidth** (CSS length): Width of the page content, default 70em.
  - **documentwidth** (CSS length): Width of the document (without sidebar), default 50em.
  - **sidebarwidth** (CSS length): Width of the sidebar, default 20em.
  - **bgcolor** (CSS color): Background color.
  - **headerbg** (CSS value for "background"): background for the header area, default a grayish gradient.
  - **footerbg** (CSS value for "background"): background for the footer area, default a light gray gradient.
  - **linkcolor** (CSS color): Body link color.
  - **headercolor1**, **headercolor2** (CSS color): colors for <h1> and <h2> headings.
  - **headerlinkcolor** (CSS color): Color for the backreference link in headings.
  - **textalign** (CSS *text-align* value): Text alignment for the body, default is `justify`.
- **nature** – A greenish theme. There are currently no options beyond *nosidebar* and *sidebarwidth*.
- **pyramid** – A theme from the Pyramid web framework project, designed by Blaise Laflamme. There are currently no options beyond *nosidebar* and *sidebarwidth*.

[6] http://jinja.pocoo.org/

- **haiku** – A theme without sidebar inspired by the Haiku OS user guide[7]. The following options are supported:
  - **full_logo** (true or false, default `False`): If this is true, the header will only show the *html_logo*. Use this for large logos. If this is false, the logo (if present) will be shown floating right, and the documentation title will be put in the header.
  - **textcolor**, **headingcolor**, **linkcolor**, **visitedlinkcolor**, **hoverlinkcolor** (CSS colors): Colors for various body elements.
- **traditional** – A theme resembling the old Python documentation. There are currently no options beyond *nosidebar* and *sidebarwidth*.
- **epub** – A theme for the epub builder. This theme tries to save visual space which is a sparse resource on ebook readers. The following options are supported:
  - **relbar1** (true or false, default `True`): If this is true, the *relbar1* block is inserted in the epub output, otherwise it is omitted.
  - **footer** (true or false, default `True`): If this is true, the *footer* block is inserted in the epub output, otherwise it is omitted.
- **bizstyle** – A simple bluish theme. The following options are supported beyond *nosidebar* and *sidebarwidth*:
  - **rightsidebar** (true or false): Put the sidebar on the right side. Defaults to `False`.

New in version 1.3: 'alabaster', 'sphinx_rtd_theme' and 'bizstyle' theme.

Changed in version 1.3: The 'default' theme has been renamed to 'classic'. 'default' is still available, however it will emit notice a recommendation that using new 'alabaster' theme.

12.3 Creating themes

As said, themes are either a directory or a zipfile (whose name is the theme name), containing the following:

- A `theme.conf` file, see below.
- HTML templates, if needed.
- A `static/` directory containing any static files that will be copied to the output static directory on build. These can be images, styles, script files.

The `theme.conf` file is in INI format [8] (readable by the standard Python `ConfigParser` module) and has the following structure:

```
[theme]
inherit = base theme
stylesheet = main CSS name
pygments_style = stylename

[options]
variable = default value
```

- The **inherit** setting gives the name of a "base theme", or `none`. The base theme will be used to locate missing templates (most themes will not have to supply most templates if they use `basic` as the base theme), its options will be inherited, and all of its static files will be used as well.

[7] http://www.haiku-os.org/docs/userguide/en/contents.html

[8] It is not an executable Python file, as opposed to `conf.py`, because that would pose an unnecessary security risk if themes are shared.

- The **stylesheet** setting gives the name of a CSS file which will be referenced in the HTML header. If you need more than one CSS file, either include one from the other via CSS' `@import`, or use a custom HTML template that adds `<link rel="stylesheet">` tags as necessary. Setting the *html_style* config value will override this setting.
- The **pygments_style** setting gives the name of a Pygments style to use for highlighting. This can be overridden by the user in the *pygments_style* config value.
- The **options** section contains pairs of variable names and default values. These options can be overridden by the user in *html_theme_options* and are accessible from all templates as `theme_<name>`.

12.3.1 Templating

The guide to templating is helpful if you want to write your own templates. What is important to keep in mind is the order in which Sphinx searches for templates:

- First, in the user's `templates_path` directories.
- Then, in the selected theme.
- Then, in its base theme, its base's base theme, etc.

When extending a template in the base theme with the same name, use the theme name as an explicit directory: `{% extends "basic/layout.html" %}`. From a user `templates_path` template, you can still use the "exclamation mark" syntax as described in the templating document.

12.3.2 Static templates

Since theme options are meant for the user to configure a theme more easily, without having to write a custom stylesheet, it is necessary to be able to template static files as well as HTML files. Therefore, Sphinx supports so-called "static templates", like this:

If the name of a file in the `static/` directory of a theme (or in the user's static path, for that matter) ends with `_t`, it will be processed by the template engine. The `_t` will be left from the final file name. For example, the *classic* theme has a file `static/classic.css_t` which uses templating to put the color options into the stylesheet. When a documentation is built with the classic theme, the output directory will contain a `_static/classic.css` file where all template tags have been processed.

CHAPTER 13

Templating

Sphinx uses the Jinja[1] templating engine for its HTML templates. Jinja is a text-based engine, and inspired by Django templates, so anyone having used Django will already be familiar with it. It also has excellent documentation for those who need to make themselves familiar with it.

13.1 Do I need to use Sphinx's templates to produce HTML?

No. You have several other options:

- You can write a `TemplateBridge` subclass that calls your template engine of choice, and set the `template_bridge` configuration value accordingly.
- You can *write a custom builder* that derives from `StandaloneHTMLBuilder` and calls your template engine of choice.
- You can use the `PickleHTMLBuilder` that produces pickle files with the page contents, and post-process them using a custom tool, or use them in your Web application.

13.2 Jinja/Sphinx Templating Primer

The default templating language in Sphinx is Jinja. It's Django/Smarty inspired and easy to understand. The most important concept in Jinja is *template inheritance*, which means that you can overwrite only specific blocks within a template, customizing it while also keeping the changes at a minimum.

To customize the output of your documentation you can override all the templates (both the layout templates and the child templates) by adding files with the same name as the original filename into the template directory of the structure the Sphinx quickstart generated for you.

Sphinx will look for templates in the folders of `templates_path` first, and if it can't find the template it's looking for there, it falls back to the selected theme's templates.

A template contains **variables**, which are replaced with values when the template is evaluated, **tags**, which control the logic of the template and **blocks** which are used for template inheritance.

Sphinx's *basic* theme provides base templates with a couple of blocks it will fill with data. These are located in the `themes/basic` subdirectory of the Sphinx installation directory, and used by all builtin Sphinx themes. Templates with the same name in the `templates_path` override templates supplied by the selected theme.

[1] http://jinja.pocoo.org

For example, to add a new link to the template area containing related links all you have to do is to add a new template called `layout.html` with the following contents:

```
{% extends "!layout.html" %}
{% block rootrellink %}
    <li><a href="http://project.invalid/">Project Homepage</a> &raquo;</li>
    {{ super() }}
{% endblock %}
```

By prefixing the name of the overridden template with an exclamation mark, Sphinx will load the layout template from the underlying HTML theme.

Important: If you override a block, call `{{ super() }}` somewhere to render the block's content in the extended template – unless you don't want that content to show up.

13.3 Working with the builtin templates

The builtin **basic** theme supplies the templates that all builtin Sphinx themes are based on. It has the following elements you can override or use:

13.3.1 Blocks

The following blocks exist in the `layout.html` template:

doctype The doctype of the output format. By default this is XHTML 1.0 Transitional as this is the closest to what Sphinx and Docutils generate and it's a good idea not to change it unless you want to switch to HTML 5 or a different but compatible XHTML doctype.

linktags This block adds a couple of `<link>` tags to the head section of the template.

extrahead This block is empty by default and can be used to add extra contents into the `<head>` tag of the generated HTML file. This is the right place to add references to JavaScript or extra CSS files.

relbar1 / relbar2 This block contains the *relation bar*, the list of related links (the parent documents on the left, and the links to index, modules etc. on the right). *relbar1* appears before the document, *relbar2* after the document. By default, both blocks are filled; to show the relbar only before the document, you would override *relbar2* like this:

```
    {% block relbar2 %}{% endblock %}
```

rootrellink / relbaritems Inside the relbar there are three sections: The *rootrellink*, the links from the documentation and the custom *relbaritems*. The *rootrellink* is a block that by default contains a list item pointing to the master document by default, the *relbaritems* is an empty block. If you override them to add extra links into the bar make sure that they are list items and end with the `reldelim1`.

document The contents of the document itself. It contains the block "body" where the individual content is put by subtemplates like `page.html`.

sidebar1 / sidebar2 A possible location for a sidebar. *sidebar1* appears before the document and is empty by default, *sidebar2* after the document and contains the default sidebar. If you want to swap the sidebar location override this and call the *sidebar* helper:

```
    {% block sidebar1 %}{{ sidebar() }}{% endblock %}
    {% block sidebar2 %}{% endblock %}
```

(The *sidebar2* location for the sidebar is needed by the `sphinxdoc.css` stylesheet, for example.)

sidebarlogo The logo location within the sidebar. Override this if you want to place some content at the top of the sidebar.

footer The block for the footer div. If you want a custom footer or markup before or after it, override this one.

The following four blocks are *only* used for pages that do not have assigned a list of custom sidebars in the `html_sidebars` config value. Their use is deprecated in favor of separate sidebar templates, which can be included via `html_sidebars`.

sidebartoc The table of contents within the sidebar.

Deprecated since version 1.0.

sidebarrel The relation links (previous, next document) within the sidebar.

Deprecated since version 1.0.

sidebarsourcelink The "Show source" link within the sidebar (normally only shown if this is enabled by `html_show_sourcelink`).

Deprecated since version 1.0.

sidebarsearch The search box within the sidebar. Override this if you want to place some content at the bottom of the sidebar.

Deprecated since version 1.0.

13.3.2 Configuration Variables

Inside templates you can set a couple of variables used by the layout template using the `{% set %}` tag:

reldelim1
The delimiter for the items on the left side of the related bar. This defaults to `' »'` Each item in the related bar ends with the value of this variable.

reldelim2
The delimiter for the items on the right side of the related bar. This defaults to `' |'`. Each item except of the last one in the related bar ends with the value of this variable.

Overriding works like this:

```
{% extends "!layout.html" %}
{% set reldelim1 = ' &gt;' %}
```

script_files
Add additional script files here, like this:

```
    {% set script_files = script_files + ["_static/myscript.js"] %}
```

css_files
Similar to `script_files`, for CSS files.

13.3.3 Helper Functions

Sphinx provides various Jinja functions as helpers in the template. You can use them to generate links or output multiply used elements.

pathto(*document*)
Return the path to a Sphinx document as a URL. Use this to refer to built documents.

pathto(*file*, *1*)
Return the path to a *file* which is a filename relative to the root of the generated output. Use this to refer to static files.

hasdoc(*document*)
Check if a document with the name *document* exists.

sidebar()
Return the rendered sidebar.

relbar()
Return the rendered relation bar.

13.3.4 Global Variables

These global variables are available in every template and are safe to use. There are more, but most of them are an implementation detail and might change in the future.

builder
The name of the builder (e.g. `html` or `htmlhelp`).

copyright
The value of *copyright*.

docstitle
The title of the documentation (the value of *html_title*), except when the "single-file" builder is used, when it is set to `None`.

embedded
True if the built HTML is meant to be embedded in some viewing application that handles navigation, not the web browser, such as for HTML help or Qt help formats. In this case, the sidebar is not included.

favicon
The path to the HTML favicon in the static path, or `''`.

file_suffix
The value of the builder's *out_suffix* attribute, i.e. the file name extension that the output files will get. For a standard HTML builder, this is usually `.html`.

has_source
True if the reST document sources are copied (if *html_copy_source* is `True`).

language
The value of *language*.

last_updated
The build date.

logo
The path to the HTML logo image in the static path, or `''`.

master_doc
The value of *master_doc*, for usage with *pathto()*.

next
The next document for the navigation. This variable is either false or has two attributes *link* and *title*. The title contains HTML markup. For example, to generate a link to the next page, you can use this snippet:

```
{% if next %}
<a href="{{ next.link|e }}">{{ next.title }}</a>
{% endif %}
```

pagename
The "page name" of the current file, i.e. either the document name if the file is generated from a reST source, or the equivalent hierarchical name relative to the output directory (`[directory/]filename_without_extension`).

parents
A list of parent documents for navigation, structured like the *next* item.

prev
Like *next*, but for the previous page.

project
The value of *project*.

release
The value of *release*.

rellinks
A list of links to put at the left side of the relbar, next to "next" and "prev". This usually contains links to the general index and other indices, such as the Python module index. If you add something yourself, it must be a tuple `(pagename, link title, accesskey, link text)`.

shorttitle
The value of *html_short_title*.

show_source
True if *html_show_sourcelink* is `True`.

sphinx_version
The version of Sphinx used to build.

style
The name of the main stylesheet, as given by the theme or *html_style*.

title
The title of the current document, as used in the `<title>` tag.

use_opensearch
The value of *html_use_opensearch*.

version
The value of *version*.

In addition to these values, there are also all **theme options** available (prefixed by `theme_`), as well as the values given by the user in *html_context*.

In documents that are created from source files (as opposed to automatically-generated files like the module index, or documents that already are in HTML form), these variables are also available:

meta
Document metadata (a dictionary), see *File-wide metadata*.

sourcename
The name of the copied source file for the current document. This is only nonempty if the *html_copy_source* value is `True`.

toc
The local table of contents for the current page, rendered as HTML bullet lists.

toctree
A callable yielding the global TOC tree containing the current page, rendered as HTML bullet lists. Optional keyword arguments:

- `collapse` (`True` by default): if true, all TOC entries that are not ancestors of the current page are collapsed
- `maxdepth` (defaults to the max depth selected in the toctree directive): the maximum depth of the tree; set it to `-1` to allow unlimited depth
- `titles_only` (`False` by default): if true, put only toplevel document titles in the tree
- `includehidden` (`False` by default): if true, the TOC tree will also contain hidden entries.

page_source_suffix
The suffix of the file that was rendered. Since we support a list of *source_suffix*, this will allow you to properly link to the original source file.

CHAPTER 14

Sphinx Extensions

Since many projects will need special features in their documentation, Sphinx allows to add "extensions" to the build process, each of which can modify almost any aspect of document processing.

This chapter describes the extensions bundled with Sphinx. For the API documentation on writing your own extension, see *Developing extensions for Sphinx*.

14.1 Builtin Sphinx extensions

These extensions are built in and can be activated by respective entries in the `extensions` configuration value:

14.1.1 `sphinx.ext.autodoc` – Include documentation from docstrings

This extension can import the modules you are documenting, and pull in documentation from docstrings in a semi-automatic way.

Note: For Sphinx (actually, the Python interpreter that executes Sphinx) to find your module, it must be importable. That means that the module or the package must be in one of the directories on `sys.path` – adapt your `sys.path` in the configuration file accordingly.

Warning: `autodoc` **imports** the modules to be documented. If any modules have side effects on import, these will be executed by `autodoc` when `sphinx-build` is run.
If you document scripts (as opposed to library modules), make sure their main routine is protected by a `if __name__ == '__main__'` condition.

For this to work, the docstrings must of course be written in correct reStructuredText. You can then use all of the usual Sphinx markup in the docstrings, and it will end up correctly in the documentation. Together with hand-written documentation, this technique eases the pain of having to maintain two locations for documentation, while at the same time avoiding auto-generated-looking pure API documentation.

If you prefer NumPy[1] or Google[2] style docstrings over reStructuredText, you can also enable the `napoleon` extension. `napoleon` is a preprocessor that converts your docstrings to correct reStructuredText before `autodoc` processes them.

[1] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt
[2] http://google-styleguide.googlecode.com/svn/trunk/pyguide.html#Comments

autodoc provides several directives that are versions of the usual *py:module*, *py:class* and so forth. On parsing time, they import the corresponding module and extract the docstring of the given objects, inserting them into the page source under a suitable *py:module*, *py:class* etc. directive.

Note: Just as *py:class* respects the current *py:module*, *autoclass* will also do so. Likewise, *automethod* will respect the current *py:class*.

.. **automodule::**
.. **autoclass::**
.. **autoexception::**

Document a module, class or exception. All three directives will by default only insert the docstring of the object itself:

```
.. autoclass:: Noodle
```

will produce source like this:

```
.. class:: Noodle

   Noodle's docstring.
```

The "auto" directives can also contain content of their own, it will be inserted into the resulting non-auto directive source after the docstring (but before any automatic member documentation).

Therefore, you can also mix automatic and non-automatic member documentation, like so:

```
.. autoclass:: Noodle
   :members: eat, slurp

   .. method:: boil(time=10)

      Boil the noodle *time* minutes.
```

Options and advanced usage

- If you want to automatically document members, there's a `members` option:

```
.. automodule:: noodle
   :members:
```

will document all module members (recursively), and

```
.. autoclass:: Noodle
   :members:
```

will document all non-private member functions and properties (that is, those whose name doesn't start with _).

For modules, `__all__` will be respected when looking for members; the order of the members will also be the order in `__all__`.

You can also give an explicit list of members; only these will then be documented:

```
.. autoclass:: Noodle
   :members: eat, slurp
```

- If you want to make the `members` option (or other flag options described below) the default, see *autodoc_default_flags*.
- Members without docstrings will be left out, unless you give the `undoc-members` flag option:

```
.. automodule:: noodle
   :members:
   :undoc-members:
```

- "Private" members (that is, those named like `_private` or `__private`) will be included if the `private-members` flag option is given.

  New in version 1.1.

- Python "special" members (that is, those named like `__special__`) will be included if the `special-members` flag option is given:

```
.. autoclass:: my.Class
   :members:
   :private-members:
   :special-members:
```

  would document both "private" and "special" members of the class.

  New in version 1.1.

  Changed in version 1.2: The option can now take arguments, i.e. the special members to document.

- For classes and exceptions, members inherited from base classes will be left out when documenting all members, unless you give the `inherited-members` flag option, in addition to `members`:

```
.. autoclass:: Noodle
   :members:
   :inherited-members:
```

  This can be combined with `undoc-members` to document *all* available members of the class or module.

  Note: this will lead to markup errors if the inherited members come from a module whose docstrings are not reST formatted.

  New in version 0.3.

- It's possible to override the signature for explicitly documented callable objects (functions, methods, classes) with the regular syntax that will override the signature gained from introspection:

```
.. autoclass:: Noodle(type)

   .. automethod:: eat(persona)
```

  This is useful if the signature from the method is hidden by a decorator.

  New in version 0.4.

- The *automodule*, *autoclass* and *autoexception* directives also support a flag option called `show-inheritance`. When given, a list of base classes will be inserted just below the class signature (when used with *automodule*, this will be inserted for every class that is documented in the module).

  New in version 0.4.

- All autodoc directives support the `noindex` flag option that has the same effect as for standard *py:function* etc. directives: no index entries are generated for the documented object (and all autodocumented members).

  New in version 0.4.

- *automodule* also recognizes the synopsis, platform and deprecated options that the standard *py:module* directive supports.

  New in version 0.5.

- *automodule* and *autoclass* also has an member-order option that can be used to override the global value of *autodoc_member_order* for one directive.

  New in version 0.6.

- The directives supporting member documentation also have a exclude-members option that can be used to exclude single member names from documentation, if all members are to be documented.

  New in version 0.6.

- In an *automodule* directive with the members option set, only module members whose __module__ attribute is equal to the module name as given to automodule will be documented. This is to prevent documentation of imported classes or functions. Set the imported-members option if you want to prevent this behavior and document all available members. Note that attributes from imported modules will not be documented, because attribute documentation is discovered by parsing the source file of the current module.

  New in version 1.2.

- Add a list of modules in the *autodoc_mock_imports* to prevent import errors to halt the building process when some external dependencies are not importable at build time.

  New in version 1.3.

.. autofunction::
.. autodata::
.. automethod::
.. autoattribute::

These work exactly like *autoclass* etc., but do not offer the options used for automatic member documentation.

autodata and *autoattribute* support the annotation option. Without this option, the representation of the object will be shown in the documentation. When the option is given without arguments, only the name of the object will be printed:

```
.. autodata:: CD_DRIVE
   :annotation:
```

You can tell sphinx what should be printed after the name:

```
.. autodata:: CD_DRIVE
   :annotation: = your CD device name
```

For module data members and class attributes, documentation can either be put into a comment with special formatting (using a `#:` to start the comment instead of just `#`), or in a docstring *after* the definition. Comments need to be either on a line of their own *before* the definition, or immediately after the assignment *on the same line*. The latter form is restricted to one line only.

This means that in the following class definition, all attributes can be autodocumented:

```
class Foo:
    """Docstring for class Foo."""

    #: Doc comment for class attribute Foo.bar.
    #: It can have multiple lines.
    bar = 1
```

```
        flox = 1.5   #: Doc comment for Foo.flox. One line only.

        baz = 2
        """Docstring for class attribute Foo.baz."""

        def __init__(self):
            #: Doc comment for instance attribute qux.
            self.qux = 3

            self.spam = 4
            """Docstring for instance attribute spam."""
```

Changed in version 0.6: *autodata* and *autoattribute* can now extract docstrings.

Changed in version 1.1: Comment docs are now allowed on the same line after an assignment.

Changed in version 1.2: *autodata* and *autoattribute* have an `annotation` option.

Note: If you document decorated functions or methods, keep in mind that autodoc retrieves its docstrings by importing the module and inspecting the `__doc__` attribute of the given function or method. That means that if a decorator replaces the decorated function with another, it must copy the original `__doc__` to the new function.

From Python 2.5, `functools.wraps()` can be used to create well-behaved decorating functions.

There are also new config values that you can set:

autoclass_content
This value selects what content will be inserted into the main body of an *autoclass* directive. The possible values are:

"class" Only the class' docstring is inserted. This is the default. You can still document `__init__` as a separate method using *automethod* or the `members` option to *autoclass*.

"both" Both the class' and the `__init__` method's docstring are concatenated and inserted.

"init" Only the `__init__` method's docstring is inserted.

New in version 0.3.

autodoc_member_order
This value selects if automatically documented members are sorted alphabetical (value `'alphabetical'`), by member type (value `'groupwise'`) or by source order (value `'bysource'`). The default is alphabetical.

Note that for source order, the module must be a Python module with the source code available.

New in version 0.6.

Changed in version 1.0: Support for `'bysource'`.

autodoc_default_flags
This value is a list of autodoc directive flags that should be automatically applied to all autodoc directives. The supported flags are `'members'`, `'undoc-members'`, `'private-members'`, `'special-members'`,`'inherited-members'` and `'show-inheritance'`.

If you set one of these flags in this config value, you can use a negated form, `'no-`*`flag`*`'`, in an autodoc directive, to disable it once. For example, if `autodoc_default_flags` is set to `['members', 'undoc-members']`, and you write a directive like this:

```
.. automodule:: foo
   :no-undoc-members:
```

the directive will be interpreted as if only `:members:` was given.

New in version 1.0.

autodoc_docstring_signature
Functions imported from C modules cannot be introspected, and therefore the signature for such functions cannot be automatically determined. However, it is an often-used convention to put the signature into the first line of the function's docstring.

If this boolean value is set to `True` (which is the default), autodoc will look at the first line of the docstring for functions and methods, and if it looks like a signature, use the line as the signature and remove it from the docstring content.

New in version 1.1.

autodoc_mock_imports
This value contains a list of modules to be mocked up. This is useful when some external dependencies are not met at build time and break the building process.

New in version 1.3.

Docstring preprocessing

autodoc provides the following additional events:

autodoc-process-docstring (*app*, *what*, *name*, *obj*, *options*, *lines*)
New in version 0.4.

Emitted when autodoc has read and processed a docstring. *lines* is a list of strings – the lines of the processed docstring – that the event handler can modify **in place** to change what Sphinx puts into the output.

Parameters

- **app** – the Sphinx application object
- **what** – the type of the object which the docstring belongs to (one of `"module"`, `"class"`, `"exception"`, `"function"`, `"method"`, `"attribute"`)
- **name** – the fully qualified name of the object
- **obj** – the object itself
- **options** – the options given to the directive: an object with attributes `inherited_members`, `undoc_members`, `show_inheritance` and `noindex` that are true if the flag option of same name was given to the auto directive
- **lines** – the lines of the docstring, see above

autodoc-process-signature (*app*, *what*, *name*, *obj*, *options*, *signature*, *return_annotation*)
New in version 0.5.

Emitted when autodoc has formatted a signature for an object. The event handler can return a new tuple `(signature, return_annotation)` to change what Sphinx puts into the output.

Parameters

- **app** – the Sphinx application object

- **what** – the type of the object which the docstring belongs to (one of "module", "class", "exception", "function", "method", "attribute")
- **name** – the fully qualified name of the object
- **obj** – the object itself
- **options** – the options given to the directive: an object with attributes inherited_members, undoc_members, show_inheritance and noindex that are true if the flag option of same name was given to the auto directive
- **signature** – function signature, as a string of the form "(parameter_1, parameter_2)", or None if introspection didn't succeed and signature wasn't specified in the directive.
- **return_annotation** – function return annotation as a string of the form " -> annotation", or None if there is no return annotation

The *sphinx.ext.autodoc* module provides factory functions for commonly needed docstring processing in event *autodoc-process-docstring*:

sphinx.ext.autodoc.**cut_lines**(*pre*, *post=0*, *what=None*)
Return a listener that removes the first *pre* and last *post* lines of every docstring. If *what* is a sequence of strings, only docstrings of a type in *what* will be processed.

Use like this (e.g. in the setup() function of conf.py):

```
from sphinx.ext.autodoc import cut_lines
app.connect('autodoc-process-docstring', cut_lines(4, what=['module']))
```

This can (and should) be used in place of automodule_skip_lines.

sphinx.ext.autodoc.**between**(*marker*, *what=None*, *keepempty=False*, *exclude=False*)
Return a listener that either keeps, or if *exclude* is True excludes, lines between lines that match the *marker* regular expression. If no line matches, the resulting docstring would be empty, so no change will be made unless *keepempty* is true.

If *what* is a sequence of strings, only docstrings of a type in *what* will be processed.

Skipping members

autodoc allows the user to define a custom method for determining whether a member should be included in the documentation by using the following event:

autodoc-skip-member(*app*, *what*, *name*, *obj*, *skip*, *options*)
New in version 0.5.

Emitted when autodoc has to decide whether a member should be included in the documentation. The member is excluded if a handler returns True. It is included if the handler returns False.

Parameters

- **app** – the Sphinx application object
- **what** – the type of the object which the docstring belongs to (one of "module", "class", "exception", "function", "method", "attribute")
- **name** – the fully qualified name of the object
- **obj** – the object itself
- **skip** – a boolean indicating if autodoc will skip this member if the user handler does not override the decision

- **options** – the options given to the directive: an object with attributes `inherited_members`, `undoc_members`, `show_inheritance` and `noindex` that are true if the flag option of same name was given to the auto directive

14.1.2 `sphinx.ext.autosummary` – Generate autodoc summaries

New in version 0.6.

This extension generates function/method/attribute summary lists, similar to those output e.g. by Epydoc and other API doc generation tools. This is especially useful when your docstrings are long and detailed, and putting each one of them on a separate page makes them easier to read.

The *sphinx.ext.autosummary* extension does this in two parts:

1. There is an *autosummary* directive for generating summary listings that contain links to the documented items, and short summary blurbs extracted from their docstrings.
2. Optionally, the convenience script **sphinx-autogen** or the new *autosummary_generate* config value can be used to generate short "stub" files for the entries listed in the *autosummary* directives. These files by default contain only the corresponding *sphinx.ext.autodoc* directive, but can be customized with templates.

`.. autosummary::`
Insert a table that contains links to documented items, and a short summary blurb (the first sentence of the docstring) for each of them.

The *autosummary* directive can also optionally serve as a *toctree* entry for the included items. Optionally, stub `.rst` files for these items can also be automatically generated.

For example,

```
.. currentmodule:: sphinx

.. autosummary::

   environment.BuildEnvironment
   util.relative_uri
```

produces a table like this:

environment.BuildEnvironment(srcdir, ...)	The environment in which the ReST files are translated.

Autosummary preprocesses the docstrings and signatures with the same *autodoc-process-docstring* and *autodoc-process-signature* hooks as *autodoc*.

Options

- If you want the *autosummary* table to also serve as a *toctree* entry, use the `toctree` option, for example:

```
.. autosummary::
   :toctree: DIRNAME

   sphinx.environment.BuildEnvironment
   sphinx.util.relative_uri
```

The `toctree` option also signals to the **sphinx-autogen** script that stub pages should be gen-

erated for the entries listed in this directive. The option accepts a directory name as an argument; **sphinx-autogen** will by default place its output in this directory. If no argument is given, output is placed in the same directory as the file that contains the directive.

- If you don't want the *autosummary* to show function signatures in the listing, include the `nosignatures` option:

```
.. autosummary::
   :nosignatures:

   sphinx.environment.BuildEnvironment
   sphinx.util.relative_uri
```

- You can specify a custom template with the `template` option. For example,

```
.. autosummary::
   :template: mytemplate.rst

   sphinx.environment.BuildEnvironment
```

would use the template `mytemplate.rst` in your *templates_path* to generate the pages for all entries listed. See *Customizing templates* below.

New in version 1.0.

sphinx-autogen – generate autodoc stub pages

The **sphinx-autogen** script can be used to conveniently generate stub documentation pages for items included in *autosummary* listings.

For example, the command

```
$ sphinx-autogen -o generated *.rst
```

will read all *autosummary* tables in the `*.rst` files that have the `:toctree:` option set, and output corresponding stub pages in directory `generated` for all documented items. The generated pages by default contain text of the form:

```
sphinx.util.relative_uri
========================

.. autofunction:: sphinx.util.relative_uri
```

If the `-o` option is not given, the script will place the output files in the directories specified in the `:toctree:` options.

Generating stub pages automatically

If you do not want to create stub pages with **sphinx-autogen**, you can also use this new config value:

autosummary_generate
: Boolean indicating whether to scan all found documents for autosummary directives, and to generate stub pages for each.

  Can also be a list of documents for which stub pages should be generated.

  The new files will be placed in the directories specified in the `:toctree:` options of the directives.

Customizing templates

New in version 1.0.

You can customize the stub page templates, in a similar way as the HTML Jinja templates, see *Templating*. (`TemplateBridge` is not supported.)

Note: If you find yourself spending much time tailoring the stub templates, this may indicate that it's a better idea to write custom narrative documentation instead.

Autosummary uses the following template files:

- `autosummary/base.rst` – fallback template
- `autosummary/module.rst` – template for modules
- `autosummary/class.rst` – template for classes
- `autosummary/function.rst` – template for functions
- `autosummary/attribute.rst` – template for class attributes
- `autosummary/method.rst` – template for class methods

The following variables available in the templates:

name
Name of the documented object, excluding the module and class parts.

objname
Name of the documented object, excluding the module parts.

fullname
Full name of the documented object, including module and class parts.

module
Name of the module the documented object belongs to.

class
Name of the class the documented object belongs to. Only available for methods and attributes.

underline
A string containing `len(full_name) * '='`.

members
List containing names of all members of the module or class. Only available for modules and classes.

functions
List containing names of "public" functions in the module. Here, "public" here means that the name does not start with an underscore. Only available for modules.

classes
List containing names of "public" classes in the module. Only available for modules.

exceptions
List containing names of "public" exceptions in the module. Only available for modules.

methods
List containing names of "public" methods in the class. Only available for classes.

attributes
List containing names of "public" attributes in the class. Only available for classes.

Note: You can use the `autosummary` directive in the stub pages. Stub pages are generated also based on these directives.

14.1.3 `sphinx.ext.doctest` – Test snippets in the documentation

This extension allows you to test snippets in the documentation in a natural way. It works by collecting specially-marked up code blocks and running them as doctest tests.

Within one document, test code is partitioned in *groups*, where each group consists of:

- zero or more *setup code* blocks (e.g. importing the module to test)
- one or more *test* blocks

When building the docs with the `doctest` builder, groups are collected for each document and run one after the other, first executing setup code blocks, then the test blocks in the order they appear in the file.

There are two kinds of test blocks:

- *doctest-style* blocks mimic interactive sessions by interleaving Python code (including the interpreter prompt) and output.
- *code-output-style* blocks consist of an ordinary piece of Python code, and optionally, a piece of output for that code.

The doctest extension provides four directives. The *group* argument is interpreted as follows: if it is empty, the block is assigned to the group named `default`. If it is `*`, the block is assigned to all groups (including the `default` group). Otherwise, it must be a comma-separated list of group names.

`.. testsetup:: [group]`
A setup code block. This code is not shown in the output for other builders, but executed before the doctests of the group(s) it belongs to.

`.. testcleanup:: [group]`
A cleanup code block. This code is not shown in the output for other builders, but executed after the doctests of the group(s) it belongs to.

New in version 1.1.

`.. doctest:: [group]`
A doctest-style code block. You can use standard `doctest` flags for controlling how actual output is compared with what you give as output. By default, these options are enabled: `ELLIPSIS` (allowing you to put ellipses in the expected output that match anything in the actual output), `IGNORE_EXCEPTION_DETAIL` (not comparing tracebacks), `DONT_ACCEPT_TRUE_FOR_1` (by default, doctest accepts "True" in the output where "1" is given – this is a relic of pre-Python 2.2 times).

This directive supports two options:

- `hide`, a flag option, hides the doctest block in other builders. By default it is shown as a highlighted doctest block.
- `options`, a string option, can be used to give a comma-separated list of doctest flags that apply to each example in the tests. (You still can give explicit flags per example, with doctest comments, but they will show up in other builders too.)

Note that like with standard doctests, you have to use `<BLANKLINE>` to signal a blank line in the expected output. The `<BLANKLINE>` is removed when building presentation output (HTML, LaTeX etc.).

Also, you can give inline doctest options, like in doctest:

```
    >>> datetime.date.now()    # doctest: +SKIP
    datetime.date(2008, 1, 1)
```

They will be respected when the test is run, but stripped from presentation output.

`.. testcode:: [group]`
A code block for a code-output-style test.

This directive supports one option:

- `hide`, a flag option, hides the code block in other builders. By default it is shown as a highlighted code block.

Note: Code in a `testcode` block is always executed all at once, no matter how many statements it contains. Therefore, output will *not* be generated for bare expressions – use `print`. Example:

```
    .. testcode::

       1+1        # this will give no output!
       print 2+2  # this will give output

    .. testoutput::

       4
```

Also, please be aware that since the doctest module does not support mixing regular output and an exception message in the same snippet, this applies to testcode/testoutput as well.

`.. testoutput:: [group]`
The corresponding output, or the exception message, for the last *`testcode`* block.

This directive supports two options:

- `hide`, a flag option, hides the output block in other builders. By default it is shown as a literal block without highlighting.
- `options`, a string option, can be used to give doctest flags (comma-separated) just like in normal doctest blocks.

Example:

```
    .. testcode::

       print 'Output     text.'

    .. testoutput::
       :hide:
       :options: -ELLIPSIS, +NORMALIZE_WHITESPACE

       Output text.
```

The following is an example for the usage of the directives. The test via *`doctest`* and the test via *`testcode`* and *`testoutput`* are equivalent.

```
The parrot module
=================

.. testsetup:: *

   import parrot
```

```
The parrot module is a module about parrots.

Doctest example:

.. doctest::

   >>> parrot.voom(3000)
   This parrot wouldn't voom if you put 3000 volts through it!

Test-Output example:

.. testcode::

   parrot.voom(3000)

This would output:

.. testoutput::

   This parrot wouldn't voom if you put 3000 volts through it!
```

There are also these config values for customizing the doctest extension:

doctest_path
A list of directories that will be added to `sys.path` when the doctest builder is used. (Make sure it contains absolute paths.)

doctest_global_setup
Python code that is treated like it were put in a `testsetup` directive for *every* file that is tested, and for every group. You can use this to e.g. import modules you will always need in your doctests.

New in version 0.6.

doctest_global_cleanup
Python code that is treated like it were put in a `testcleanup` directive for *every* file that is tested, and for every group. You can use this to e.g. remove any temporary files that the tests leave behind.

New in version 1.1.

doctest_test_doctest_blocks
If this is a nonempty string (the default is `'default'`), standard reST doctest blocks will be tested too. They will be assigned to the group name given.

reST doctest blocks are simply doctests put into a paragraph of their own, like so:

```
Some documentation text.

>>> print 1
1

Some more documentation text.
```

(Note that no special `::` is used to introduce a doctest block; docutils recognizes them from the leading `>>>`. Also, no additional indentation is used, though it doesn't hurt.)

If this value is left at its default value, the above snippet is interpreted by the doctest builder exactly like the following:

```
Some documentation text.

.. doctest::
```

```
    >>> print 1
    1

Some more documentation text.
```

This feature makes it easy for you to test doctests in docstrings included with the *autodoc* extension without marking them up with a special directive.

Note though that you can't have blank lines in reST doctest blocks. They will be interpreted as one block ending and another one starting. Also, removal of `<BLANKLINE>` and `# doctest:` options only works in *doctest* blocks, though you may set *trim_doctest_flags* to achieve that in all code blocks with Python console content.

14.1.4 `sphinx.ext.intersphinx` – Link to other projects' documentation

New in version 0.5.

This extension can generate automatic links to the documentation of objects in other projects.

Usage is simple: whenever Sphinx encounters a cross-reference that has no matching target in the current documentation set, it looks for targets in the documentation sets configured in *intersphinx_mapping*. A reference like `:py:class:`zipfile.ZipFile`` can then link to the Python documentation for the ZipFile class, without you having to specify where it is located exactly.

When using the "new" format (see below), you can even force lookup in a foreign set by prefixing the link target appropriately. A link like `:ref:`comparison manual <python:comparisons>`` will then link to the label "comparisons" in the doc set "python", if it exists.

Behind the scenes, this works as follows:

- Each Sphinx HTML build creates a file named `objects.inv` that contains a mapping from object names to URIs relative to the HTML set's root.
- Projects using the Intersphinx extension can specify the location of such mapping files in the *intersphinx_mapping* config value. The mapping will then be used to resolve otherwise missing references to objects into links to the other documentation.
- By default, the mapping file is assumed to be at the same location as the rest of the documentation; however, the location of the mapping file can also be specified individually, e.g. if the docs should be buildable without Internet access.

To use intersphinx linking, add `'sphinx.ext.intersphinx'` to your *extensions* config value, and use these new config values to activate linking:

intersphinx_mapping

This config value contains the locations and names of other projects that should be linked to in this documentation.

Relative local paths for target locations are taken as relative to the base of the built documentation, while relative local paths for inventory locations are taken as relative to the source directory.

When fetching remote inventory files, proxy settings will be read from the `$HTTP_PROXY` environment variable.

Old format for this config value

This is the format used before Sphinx 1.0. It is still recognized.

A dictionary mapping URIs to either `None` or an URI. The keys are the base URI of the foreign Sphinx documentation sets and can be local paths or HTTP URIs. The values indicate where the inventory

file can be found: they can be `None` (at the same location as the base URI) or another local or HTTP URI.

New format for this config value

New in version 1.0.

A dictionary mapping unique identifiers to a tuple `(target, inventory)`. Each `target` is the base URI of a foreign Sphinx documentation set and can be a local path or an HTTP URI. The `inventory` indicates where the inventory file can be found: it can be `None` (at the same location as the base URI) or another local or HTTP URI.

The unique identifier can be used to prefix cross-reference targets, so that it is clear which intersphinx set the target belongs to. A link like `:ref:'comparison manual <python:comparisons>'` will link to the label "comparisons" in the doc set "python", if it exists.

Example

To add links to modules and objects in the Python standard library documentation, use:

```
intersphinx_mapping = {'python': ('https://docs.python.org/3.4', None)}
```

This will download the corresponding `objects.inv` file from the Internet and generate links to the pages under the given URI. The downloaded inventory is cached in the Sphinx environment, so it must be re-downloaded whenever you do a full rebuild.

A second example, showing the meaning of a non-`None` value of the second tuple item:

```
intersphinx_mapping = {'python': ('https://docs.python.org/3.4',
                                  'python-inv.txt')}
```

This will read the inventory from `python-inv.txt` in the source directory, but still generate links to the pages under `https://docs.python.org/3.4`. It is up to you to update the inventory file as new objects are added to the Python documentation.

Multiple target for the inventory

New in version 1.3.

Alternative files can be specified for each inventory. One can give a tuple for the second inventory tuple item as shown in the following example. This will read the inventory iterating through the (second) tuple items until the first successful fetch. The primary use case for this to specify mirror sites for server downtime of the primary inventory:

```
intersphinx_mapping = {'python': ('https://docs.python.org/3.4',
                                  (None, 'python-inv.txt'))}
```

intersphinx_cache_limit
The maximum number of days to cache remote inventories. The default is `5`, meaning five days. Set this to a negative value to cache inventories for unlimited time.

14.1.5 Math support in Sphinx

New in version 0.5.

Since mathematical notation isn't natively supported by HTML in any way, Sphinx supports math in documentation with several extensions.

The basic math support is contained in *sphinx.ext.mathbase*. Other math support extensions should, if possible, reuse that support too.

Note: *mathbase* is not meant to be added to the *extensions* config value, instead, use either *sphinx.ext.pngmath* or *sphinx.ext.mathjax* as described below.

The input language for mathematics is LaTeX markup. This is the de-facto standard for plain-text math notation and has the added advantage that no further translation is necessary when building LaTeX output.

Keep in mind that when you put math markup in **Python docstrings** read by *autodoc*, you either have to double all backslashes, or use Python raw strings (`r"raw"`).

mathbase defines these new markup elements:

:math:
Role for inline math. Use like this:

```
Since Pythagoras, we know that :math:`a^2 + b^2 = c^2`.
```

.. math::
Directive for displayed math (math that takes the whole line for itself).

The directive supports multiple equations, which should be separated by a blank line:

```
.. math::

   (a + b)^2 = a^2 + 2ab + b^2

   (a - b)^2 = a^2 - 2ab + b^2
```

In addition, each single equation is set within a `split` environment, which means that you can have multiple aligned lines in an equation, aligned at `&` and separated by `\\`:

```
.. math::

   (a + b)^2  &=  (a + b)(a + b) \\
              &=  a^2 + 2ab + b^2
```

For more details, look into the documentation of the AmSMath LaTeX package[3].

When the math is only one line of text, it can also be given as a directive argument:

```
.. math:: (a + b)^2 = a^2 + 2ab + b^2
```

Normally, equations are not numbered. If you want your equation to get a number, use the `label` option. When given, it selects an internal label for the equation, by which it can be cross-referenced, and causes an equation number to be issued. See `eqref` for an example. The numbering style depends on the output format.

There is also an option `nowrap` that prevents any wrapping of the given math in a math environment. When you give this option, you must make sure yourself that the math is properly set up. For example:

```
.. math::
   :nowrap:

   \begin{eqnarray}
      y    & = & ax^2 + bx + c \\
      f(x) & = & x^2 + 2xy + y^2
   \end{eqnarray}
```

[3] http://www.ams.org/publications/authors/tex/amslatex

:eq:
Role for cross-referencing equations via their label. This currently works only within the same document. Example:

```
.. math:: e^{i\pi} + 1 = 0
   :label: euler

Euler's identity, equation :eq:`euler`, was elected one of the most
beautiful mathematical formulas.
```

sphinx.ext.pngmath – Render math as PNG images

This extension renders math via LaTeX and dvipng[4] into PNG images. This of course means that the computer where the docs are built must have both programs available.

There are various config values you can set to influence how the images are built:

pngmath_latex
The command name with which to invoke LaTeX. The default is `'latex'`; you may need to set this to a full path if `latex` is not in the executable search path.

Since this setting is not portable from system to system, it is normally not useful to set it in `conf.py`; rather, giving it on the **sphinx-build** command line via the `-D` option should be preferable, like this:

```
sphinx-build -b html -D pngmath_latex=C:\tex\latex.exe . _build/html
```

Changed in version 0.5.1: This value should only contain the path to the latex executable, not further arguments; use *pngmath_latex_args* for that purpose.

pngmath_dvipng
The command name with which to invoke `dvipng`. The default is `'dvipng'`; you may need to set this to a full path if `dvipng` is not in the executable search path.

pngmath_latex_args
Additional arguments to give to latex, as a list. The default is an empty list.

New in version 0.5.1.

pngmath_latex_preamble
Additional LaTeX code to put into the preamble of the short LaTeX files that are used to translate the math snippets. This is empty by default. Use it e.g. to add more packages whose commands you want to use in the math.

pngmath_dvipng_args
Additional arguments to give to dvipng, as a list. The default value is `['-gamma', '1.5', '-D', '110', '-bg', 'Transparent']` which makes the image a bit darker and larger then it is by default, and produces PNGs with a transparent background.

Changed in version 1.2: Now includes `-bg Transparent` by default.

pngmath_use_preview
`dvipng` has the ability to determine the "depth" of the rendered text: for example, when typesetting a fraction inline, the baseline of surrounding text should not be flush with the bottom of the image, rather the image should extend a bit below the baseline. This is what TeX calls "depth". When this is enabled, the images put into the HTML document will get a `vertical-align` style that correctly aligns the baselines.

[4] http://savannah.nongnu.org/projects/dvipng/

Unfortunately, this only works when the preview-latex package[5] is installed. Therefore, the default for this option is `False`.

pngmath_add_tooltips
Default: `True`. If false, do not add the LaTeX code as an "alt" attribute for math images.

New in version 1.1.

sphinx.ext.mathjax – Render math via JavaScript

New in version 1.1.

This extension puts math as-is into the HTML files. The JavaScript package MathJax[6] is then loaded and transforms the LaTeX markup to readable math live in the browser.

Because MathJax (and the necessary fonts) is very large, it is not included in Sphinx.

mathjax_path
The path to the JavaScript file to include in the HTML files in order to load MathJax.

The default is the `http://` URL that loads the JS files from the MathJax CDN[7]. If you want MathJax to be available offline, you have to download it and set this value to a different path.

The path can be absolute or relative; if it is relative, it is relative to the `_static` directory of the built docs.

For example, if you put MathJax into the static path of the Sphinx docs, this value would be `MathJax/MathJax.js`. If you host more than one Sphinx documentation set on one server, it is advisable to install MathJax in a shared location.

You can also give a full `http://` URL different from the CDN URL.

sphinx.ext.jsmath – Render math via JavaScript

This extension works just as the MathJax extension does, but uses the older package jsMath[8]. It provides this config value:

jsmath_path
The path to the JavaScript file to include in the HTML files in order to load JSMath. There is no default.

The path can be absolute or relative; if it is relative, it is relative to the `_static` directory of the built docs.

For example, if you put JSMath into the static path of the Sphinx docs, this value would be `jsMath/easy/load.js`. If you host more than one Sphinx documentation set on one server, it is advisable to install jsMath in a shared location.

14.1.6 sphinx.ext.graphviz – Add Graphviz graphs

New in version 0.6.

This extension allows you to embed Graphviz[9] graphs in your documents.

It adds these directives:

[5] http://www.gnu.org/software/auctex/preview-latex.html
[6] http://www.mathjax.org/
[7] http://docs.mathjax.org/en/latest/start.html
[8] http://www.math.union.edu/ dpvc/jsmath/
[9] http://graphviz.org/

.. graphviz::

Directive to embed graphviz code. The input code for `dot` is given as the content. For example:

```
.. graphviz::

   digraph foo {
      "bar" -> "baz";
   }
```

In HTML output, the code will be rendered to a PNG or SVG image (see *graphviz_output_format*). In LaTeX output, the code will be rendered to an embeddable PDF file.

You can also embed external dot files, by giving the file name as an argument to *graphviz* and no additional content:

```
.. graphviz:: external.dot
```

As for all file references in Sphinx, if the filename is absolute, it is taken as relative to the source directory.

Changed in version 1.1: Added support for external files.

.. graph::

Directive for embedding a single undirected graph. The name is given as a directive argument, the contents of the graph are the directive content. This is a convenience directive to generate `graph <name> { <content> }`.

For example:

```
.. graph:: foo

   "bar" -- "baz";
```

Note: The graph name is passed unchanged to Graphviz. If it contains non-alphanumeric characters (e.g. a dash), you will have to double-quote it.

.. digraph::

Directive for embedding a single directed graph. The name is given as a directive argument, the contents of the graph are the directive content. This is a convenience directive to generate `digraph <name> { <content> }`.

For example:

```
.. digraph:: foo

   "bar" -> "baz" -> "quux";
```

New in version 1.0: All three directives support an `alt` option that determines the image's alternate text for HTML output. If not given, the alternate text defaults to the graphviz code.

New in version 1.1: All three directives support an `inline` flag that controls paragraph breaks in the output. When set, the graph is inserted into the current paragraph. If the flag is not given, paragraph breaks are introduced before and after the image (the default).

New in version 1.1: All three directives support a `caption` option that can be used to give a caption to the diagram. Naturally, diagrams marked as "inline" cannot have a caption.

There are also these new config values:

graphviz_dot
The command name with which to invoke `dot`. The default is `'dot'`; you may need to set this to a full path if `dot` is not in the executable search path.

Since this setting is not portable from system to system, it is normally not useful to set it in `conf.py`; rather, giving it on the **sphinx-build** command line via the `-D` option should be preferable, like this:

```
sphinx-build -b html -D graphviz_dot=C:\graphviz\bin\dot.exe . _build/html
```

graphviz_dot_args
Additional command-line arguments to give to dot, as a list. The default is an empty list. This is the right place to set global graph, node or edge attributes via dot's `-G`, `-N` and `-E` options.

graphviz_output_format
The output format for Graphviz when building HTML files. This must be either `'png'` or `'svg'`; the default is `'png'`.

New in version 1.0: Previously, output always was PNG.

14.1.7 `sphinx.ext.inheritance_diagram` – Include inheritance diagrams

New in version 0.6.

This extension allows you to include inheritance diagrams, rendered via the *Graphviz extension*.

It adds this directive:

.. inheritance-diagram::
This directive has one or more arguments, each giving a module or class name. Class names can be unqualified; in that case they are taken to exist in the currently described module (see *py:module*).

For each given class, and each class in each given module, the base classes are determined. Then, from all classes and their base classes, a graph is generated which is then rendered via the graphviz extension to a directed graph.

This directive supports an option called `parts` that, if given, must be an integer, advising the directive to remove that many parts of module names from the displayed names. (For example, if all your class names start with `lib.`, you can give `:parts: 1` to remove that prefix from the displayed node names.)

It also supports a `private-bases` flag option; if given, private base classes (those whose name starts with _) will be included.

Changed in version 1.1: Added `private-bases` option; previously, all bases were always included.

New config values are:

inheritance_graph_attrs
A dictionary of graphviz graph attributes for inheritance diagrams.

For example:

```
inheritance_graph_attrs = dict(rankdir="LR", size='"6.0, 8.0"',
                               fontsize=14, ratio='compress')
```

inheritance_node_attrs
A dictionary of graphviz node attributes for inheritance diagrams.

For example:

```
    inheritance_node_attrs = dict(shape='ellipse', fontsize=14, height=0.75,
                                  color='dodgerblue1', style='filled')
```

inheritance_edge_attrs
A dictionary of graphviz edge attributes for inheritance diagrams.

14.1.8 sphinx.ext.ifconfig – Include content based on configuration

This extension is quite simple, and features only one directive:

.. ifconfig::
Include content of the directive only if the Python expression given as an argument is `True`, evaluated in the namespace of the project's configuration (that is, all registered variables from `conf.py` are available).

For example, one could write

```
    .. ifconfig:: releaselevel in ('alpha', 'beta', 'rc')

       This stuff is only included in the built docs for unstable versions.
```

To make a custom config value known to Sphinx, use *add_config_value()* in the setup function in `conf.py`, e.g.:

```
    def setup(app):
        app.add_config_value('releaselevel', '', 'env')
```

The second argument is the default value, the third should always be `'env'` for such values (it selects if Sphinx re-reads the documents if the value changes).

14.1.9 sphinx.ext.coverage – Collect doc coverage stats

This extension features one additional builder, the *CoverageBuilder*.

class sphinx.ext.coverage.**CoverageBuilder**
To use this builder, activate the coverage extension in your configuration file and give `-b coverage` on the command line.

Todo

Write this section.

Several new configuration values can be used to specify what the builder should check:

coverage_ignore_modules

coverage_ignore_functions

coverage_ignore_classes

coverage_c_path

coverage_c_regexes

coverage_ignore_c_items

coverage_write_headline
Set to `False` to not write headlines.

New in version 1.1.

coverage_skip_undoc_in_source
Skip objects that are not documented in the source with a docstring. `False` by default.

New in version 1.1.

14.1.10 sphinx.ext.todo – Support for todo items

Module author: Daniel Bültmann

New in version 0.5.

There are two additional directives when using this extension:

.. todo::
Use this directive like, for example, *note*.

It will only show up in the output if *todo_include_todos* is `True`.

.. todolist::
This directive is replaced by a list of all todo directives in the whole documentation, if *todo_include_todos* is `True`.

There is also an additional config value:

todo_include_todos
If this is `True`, *todo* and *todolist* produce output, else they produce nothing. The default is `False`.

14.1.11 sphinx.ext.extlinks – Markup to shorten external links

Module author: Georg Brandl

New in version 1.0.

This extension is meant to help with the common pattern of having many external links that point to URLs on one and the same site, e.g. links to bug trackers, version control web interfaces, or simply subpages in other websites. It does so by providing aliases to base URLs, so that you only need to give the subpage name when creating a link.

Let's assume that you want to include many links to issues at the Sphinx tracker, at `https://github.com/sphinx-doc/sphinx/issues/num`. Typing this URL again and again is tedious, so you can use *extlinks* to avoid repeating yourself.

The extension adds one new config value:

extlinks
This config value must be a dictionary of external sites, mapping unique short alias names to a base URL and a *prefix*. For example, to create an alias for the above mentioned issues, you would add

```
extlinks = {'issue': ('https://github.com/sphinx-doc/sphinx/issues/%s',
                      'issue ')}
```

Now, you can use the alias name as a new role, e.g. `:issue:`123``. This then inserts a link to https://github.com/sphinx-doc/sphinx/issues/123. As you can see, the target given in the role is substituted in the base URL in the place of `%s`.

The link *caption* depends on the second item in the tuple, the *prefix*:

- If the prefix is `None`, the link caption is the full URL.
- If the prefix is the empty string, the link caption is the partial URL given in the role content (`123` in this case.)

- If the prefix is a non-empty string, the link caption is the partial URL, prepended by the prefix – in the above example, the link caption would be `issue 123`.

You can also use the usual "explicit title" syntax supported by other roles that generate links, i.e. `:issue:`this issue <123>``. In this case, the *prefix* is not relevant.

Note: Since links are generated from the role in the reading stage, they appear as ordinary links to e.g. the `linkcheck` builder.

14.1.12 `sphinx.ext.viewcode` – Add links to highlighted source code

Module author: Georg Brandl

New in version 1.0.

This extension looks at your Python object descriptions (`.. class::`, `.. function::` etc.) and tries to find the source files where the objects are contained. When found, a separate HTML page will be output for each module with a highlighted version of the source code, and a link will be added to all object descriptions that leads to the source code of the described object. A link back from the source to the description will also be inserted.

There are currently no configuration values for this extension; you just need to add `'sphinx.ext.viewcode'` to your *extensions* value for it to work.

There is also an additional config value:

viewcode_import
If this is `True`, viewcode extension will follow alias objects that imported from another module such as functions, classes and attributes. As side effects, this option else they produce nothing. The default is `True`.

> **Warning:** *viewcode_import* **imports** the modules to be followed real location. If any modules have side effects on import, these will be executed by `viewcode` when `sphinx-build` is run. If you document scripts (as opposed to library modules), make sure their main routine is protected by a `if __name__ == '__main__'` condition.

New in version 1.3.

14.1.13 `sphinx.ext.linkcode` – Add external links to source code

Module author: Pauli Virtanen

New in version 1.2.

This extension looks at your object descriptions (`.. class::`, `.. function::` etc.) and adds external links to code hosted somewhere on the web. The intent is similar to the `sphinx.ext.viewcode` extension, but assumes the source code can be found somewhere on the Internet.

In your configuration, you need to specify a *linkcode_resolve* function that returns an URL based on the object.

linkcode_resolve
This is a function `linkcode_resolve(domain, info)`, which should return the URL to source code corresponding to the object in given domain with given information.

The function should return `None` if no link is to be added.

The argument `domain` specifies the language domain the object is in. `info` is a dictionary with the following keys guaranteed to be present (dependent on the domain):

- `py`: `module` (name of the module), `fullname` (name of the object)
- `c`: `names` (list of names for the object)
- `cpp`: `names` (list of names for the object)
- `javascript`: `object` (name of the object), `fullname` (name of the item)

Example:

```
    def linkcode_resolve(domain, info):
        if domain != 'py':
            return None
        if not info['module']:
            return None
        filename = info['module'].replace('.', '/')
        return "http://somesite/sourcerepo/%s.py" % filename
```

14.1.14 `sphinx.ext.napoleon` – Support for NumPy and Google style docstrings

Module author: Rob Ruana

New in version 1.3.

Napoleon - *Marching toward legible docstrings*

Are you tired of writing docstrings that look like this:

```
:param path: The path of the file to wrap
:type path: str
:param field_storage: The :class:`FileStorage` instance to wrap
:type field_storage: FileStorage
:param temporary: Whether or not to delete the file when the File
   instance is destructed
:type temporary: bool
:returns: A buffered writable file descriptor
:rtype: BufferedFileStorage
```

ReStructuredText[10] is great, but it creates visually dense, hard to read docstrings[11]. Compare the jumble above to the same thing rewritten according to the Google Python Style Guide[12]:

```
Args:
    path (str): The path of the file to wrap
    field_storage (FileStorage): The :class:`FileStorage` instance to wrap
    temporary (bool): Whether or not to delete the file when the File
       instance is destructed

Returns:
    BufferedFileStorage: A buffered writable file descriptor
```

Much more legible, no?

[10] http://docutils.sourceforge.net/rst.html
[11] http://www.python.org/dev/peps/pep-0287/
[12] http://google.github.io/styleguide/pyguide.html

Napoleon is a Sphinx Extensions that enables Sphinx to parse both NumPy[13] and Google[14] style docstrings - the style recommended by Khan Academy[15].

Napoleon is a pre-processor that parses NumPy[16] and Google[17] style docstrings and converts them to reStructuredText before Sphinx attempts to parse them. This happens in an intermediate step while Sphinx is processing the documentation, so it doesn't modify any of the docstrings in your actual source code files.

Getting Started

1. After setting up Sphinx to build your docs, enable napoleon in the Sphinx *conf.py* file:

```
# conf.py

# Add autodoc and napoleon to the extensions list
extensions = ['sphinx.ext.autodoc', 'sphinx.ext.napoleon']
```

2. Use *sphinx-apidoc* to build your API documentation:

```
$ sphinx-apidoc -f -o docs/source projectdir
```

Docstrings

Napoleon interprets every docstring that `autodoc` can find, including docstrings on: `modules`, `classes`, `attributes`, `methods`, `functions`, and `variables`. Inside each docstring, specially formatted *Sections* are parsed and converted to reStructuredText.

All standard reStructuredText formatting still works as expected.

Docstring Sections

All of the following section headers are supported:

- `Args` *(alias of Parameters)*
- `Arguments` *(alias of Parameters)*
- `Attributes`
- `Example`
- `Examples`
- `Keyword Args` *(alias of Keyword Arguments)*
- `Keyword Arguments`
- `Methods`
- `Note`
- `Notes`
- `Other Parameters`

[13] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt
[14] http://google.github.io/styleguide/pyguide.html#Comments
[15] https://github.com/Khan/style-guides/blob/master/style/python.md#docstrings
[16] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt
[17] http://google.github.io/styleguide/pyguide.html#Comments

- `Parameters`
- `Return` *(alias of Returns)*
- `Returns`
- `Raises`
- `References`
- `See Also`
- `Warning`
- `Warnings` *(alias of Warning)*
- `Warns`
- `Yield` *(alias of Yields)*
- `Yields`

Google vs NumPy

Napoleon supports two styles of docstrings: Google[18] and NumPy[19]. The main difference between the two styles is that Google uses indention to separate sections, whereas NumPy uses underlines.

Google style:

```
def func(arg1, arg2):
    """Summary line.

    Extended description of function.

    Args:
        arg1 (int): Description of arg1
        arg2 (str): Description of arg2

    Returns:
        bool: Description of return value

    """
    return True
```

NumPy style:

```
def func(arg1, arg2):
    """Summary line.

    Extended description of function.

    Parameters
    ----------
    arg1 : int
        Description of arg1
    arg2 : str
        Description of arg2

    Returns
```

[18] http://google.github.io/styleguide/pyguide.html#Comments
[19] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

```
    -------
    bool
        Description of return value

    """
    return True
```

NumPy style tends to require more vertical space, whereas Google style tends to use more horizontal space. Google style tends to be easier to read for short and simple docstrings, whereas NumPy style tends be easier to read for long and in-depth docstrings.

The Khan Academy[20] recommends using Google style.

The choice between styles is largely aesthetic, but the two styles should not be mixed. Choose one style for your project and be consistent with it.

See also:

For complete examples:

- example_google
- example_numpy

For Python type annotations, see PEP 484[21].

Configuration

Listed below are all the settings used by napoleon and their default values. These settings can be changed in the Sphinx *conf.py* file. Make sure that both "sphinx.ext.autodoc" and "sphinx.ext.napoleon" are enabled in *conf.py*:

```
# conf.py

# Add any Sphinx extension module names here, as strings
extensions = ['sphinx.ext.autodoc', 'sphinx.ext.napoleon']

# Napoleon settings
napoleon_google_docstring = True
napoleon_numpy_docstring = True
napoleon_include_private_with_doc = False
napoleon_include_special_with_doc = True
napoleon_use_admonition_for_examples = False
napoleon_use_admonition_for_notes = False
napoleon_use_admonition_for_references = False
napoleon_use_ivar = False
napoleon_use_param = True
napoleon_use_rtype = True
```

napoleon_google_docstring
: True to parse Google style[22] docstrings. False to disable support for Google style docstrings. *Defaults to True.*

napoleon_numpy_docstring
: True to parse NumPy style[23] docstrings. False to disable support for NumPy style docstrings. *Defaults to True.*

[20] https://github.com/Khan/style-guides/blob/master/style/python.md#docstrings
[21] https://www.python.org/dev/peps/pep-0484/
[22] http://google-styleguide.googlecode.com/svn/trunk/pyguide.html
[23] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

napoleon_include_private_with_doc
True to include private members (like `_membername`) with docstrings in the documentation. False to fall back to Sphinx's default behavior. *Defaults to False.*

If True:

```
def _included(self):
    """
    This will be included in the docs because it has a docstring
    """
    pass

def _skipped(self):
    # This will NOT be included in the docs
    pass
```

napoleon_include_special_with_doc
True to include special members (like `__membername__`) with docstrings in the documentation. False to fall back to Sphinx's default behavior. *Defaults to True.*

If True:

```
def __str__(self):
    """
    This will be included in the docs because it has a docstring
    """
    return unicode(self).encode('utf-8')

def __unicode__(self):
    # This will NOT be included in the docs
    return unicode(self.__class__.__name__)
```

napoleon_use_admonition_for_examples
True to use the `.. admonition::` directive for the **Example** and **Examples** sections. False to use the `.. rubric::` directive instead. One may look better than the other depending on what HTML theme is used. *Defaults to False.*

This NumPy style[24] snippet will be converted as follows:

```
Example
-------
This is just a quick example
```

If True:

```
.. admonition:: Example

   This is just a quick example
```

If False:

```
.. rubric:: Example

This is just a quick example
```

napoleon_use_admonition_for_notes
True to use the `.. admonition::` directive for **Notes** sections. False to use the `.. rubric::` directive instead. *Defaults to False.*

[24] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

Note: The singular **Note** section will always be converted to a `.. note::` directive.

See also:

`napoleon_use_admonition_for_examples`

napoleon_use_admonition_for_references
True to use the `.. admonition::` directive for **References** sections. False to use the `.. rubric::` directive instead. *Defaults to False.*

See also:

`napoleon_use_admonition_for_examples`

napoleon_use_ivar
True to use the `:ivar:` role for instance variables. False to use the `.. attribute::` directive instead. *Defaults to False.*

This NumPy style[25] snippet will be converted as follows:

```
Attributes
----------
attr1 : int
    Description of `attr1`
```

If True:

```
:ivar attr1: Description of `attr1`
:vartype attr1: int
```

If False:

```
.. attribute:: attr1

   *int*

   Description of `attr1`
```

napoleon_use_param
True to use a `:param:` role for each function parameter. False to use a single `:parameters:` role for all the parameters. *Defaults to True.*

This NumPy style[26] snippet will be converted as follows:

```
Parameters
----------
arg1 : str
    Description of `arg1`
arg2 : int, optional
    Description of `arg2`, defaults to 0
```

If True:

```
:param arg1: Description of `arg1`
:type arg1: str
:param arg2: Description of `arg2`, defaults to 0
:type arg2: int, optional
```

If False:

[25] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt
[26] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

```
    :parameters: * **arg1** (*str*) --
                   Description of `arg1`
                 * **arg2** (*int, optional*) --
                   Description of `arg2`, defaults to 0
```

napoleon_use_rtype
: True to use the `:rtype:` role for the return type. False to output the return type inline with the description. *Defaults to True.*

This NumPy style[27] snippet will be converted as follows:

```
    Returns
    -------
    bool
        True if successful, False otherwise
```

If True:

```
    :returns: True if successful, False otherwise
    :rtype: bool
```

If False:

```
    :returns: *bool* -- True if successful, False otherwise
```

14.2 Third-party extensions

You can find several extensions contributed by users in the Sphinx Contrib[28] repository. It is open for anyone who wants to maintain an extension publicly; just send a short message asking for write permissions.

There are also several extensions hosted elsewhere. The Sphinx extension survey[29] contains a comprehensive list.

If you write an extension that you think others will find useful or you think should be included as a part of Sphinx, please write to the project mailing list (join here[30]).

14.2.1 Where to put your own extensions?

Extensions local to a project should be put within the project's directory structure. Set Python's module search path, `sys.path`, accordingly so that Sphinx can find them. E.g., if your extension `foo.py` lies in the `exts` subdirectory of the project root, put into `conf.py`:

```
import sys, os

sys.path.append(os.path.abspath('exts'))

extensions = ['foo']
```

You can also install extensions anywhere else on `sys.path`, e.g. in the `site-packages` directory.

[27] https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt
[28] https://bitbucket.org/birkenfeld/sphinx-contrib
[29] http://sphinxext-survey.readthedocs.org/en/latest/
[30] https://groups.google.com/group/sphinx-dev

CHAPTER 15

Developing extensions for Sphinx

Since many projects will need special features in their documentation, Sphinx is designed to be extensible on several levels.

This is what you can do in an extension: First, you can add new *builder*s to support new output formats or actions on the parsed documents. Then, it is possible to register custom reStructuredText roles and directives, extending the markup. And finally, there are so-called "hook points" at strategic places throughout the build process, where an extension can register a hook and run specialized code.

An extension is simply a Python module. When an extension is loaded, Sphinx imports this module and executes its `setup()` function, which in turn notifies Sphinx of everything the extension offers – see the extension tutorial for examples.

The configuration file itself can be treated as an extension if it contains a `setup()` function. All other extensions to load must be listed in the *`extensions`* configuration value.

15.1 Extension metadata

New in version 1.3.

The `setup()` function can return a dictionary. This is treated by Sphinx as metadata of the extension. Metadata keys currently recognized are:

- `'version'`: a string that identifies the extension version. It is used for extension version requirement checking (see *`needs_extensions`*) and informational purposes. If not given, `"unknown version"` is substituted.
- `'parallel_read_safe'`: a boolean that specifies if parallel reading of source files can be used when the extension is loaded. It defaults to `False`, i.e. you have to explicitly specify your extension to be parallel-read-safe after checking that it is.
- `'parallel_write_safe'`: a boolean that specifies if parallel writing of output files can be used when the extension is loaded. Since extensions usually don't negatively influence the process, this defaults to `True`.

15.2 APIs used for writing extensions

15.2.1 Tutorial: Writing a simple extension

This section is intended as a walkthrough for the creation of custom extensions. It covers the basics of writing and activating an extension, as well as commonly used features of extensions.

As an example, we will cover a "todo" extension that adds capabilities to include todo entries in the documentation, and to collect these in a central place. (A similar "todo" extension is distributed with Sphinx.)

Important objects

There are several key objects whose API you will use while writing an extension. These are:

Application The application object (usually called `app`) is an instance of *`Sphinx`*. It controls most high-level functionality, such as the setup of extensions, event dispatching and producing output (logging).

If you have the environment object, the application is available as `env.app`.

Environment The build environment object (usually called `env`) is an instance of *`BuildEnvironment`*. It is responsible for parsing the source documents, stores all metadata about the document collection and is serialized to disk after each build.

Its API provides methods to do with access to metadata, resolving references, etc. It can also be used by extensions to cache information that should persist for incremental rebuilds.

If you have the application or builder object, the environment is available as `app.env` or `builder.env`.

Builder The builder object (usually called `builder`) is an instance of a specific subclass of *`Builder`*. Each builder class knows how to convert the parsed documents into an output format, or otherwise process them (e.g. check external links).

If you have the application object, the builder is available as `app.builder`.

Config The config object (usually called `config`) provides the values of configuration values set in `conf.py` as attributes. It is an instance of *`Config`*.

The config is available as `app.config` or `env.config`.

Build Phases

One thing that is vital in order to understand extension mechanisms is the way in which a Sphinx project is built: this works in several phases.

Phase 0: Initialization

In this phase, almost nothing of interest to us happens. The source directory is searched for source files, and extensions are initialized. Should a stored build environment exist, it is loaded, otherwise a new one is created.

Phase 1: Reading

In Phase 1, all source files (and on subsequent builds, those that are new or changed) are read and parsed. This is the phase where directives and roles are encountered by docutils, and the corresponding code is executed. The output of this phase is a *doctree* for each source file; that is a tree of docutils nodes. For document elements that aren't fully known until all existing files are read, temporary nodes are created.

There are nodes provided by docutils, which are documented in the docutils documentation[1]. Additional nodes are provided by Sphinx and *documented here*.

During reading, the build environment is updated with all meta- and cross reference data of the read documents, such as labels, the names of headings, described Python objects and index entries. This will later be used to replace the temporary nodes.

The parsed doctrees are stored on the disk, because it is not possible to hold all of them in memory.

Phase 2: Consistency checks

Some checking is done to ensure no surprises in the built documents.

Phase 3: Resolving

Now that the metadata and cross-reference data of all existing documents is known, all temporary nodes are replaced by nodes that can be converted into output. For example, links are created for object references that exist, and simple literal nodes are created for those that don't.

Phase 4: Writing

This phase converts the resolved doctrees to the desired output format, such as HTML or LaTeX. This happens via a so-called docutils writer that visits the individual nodes of each doctree and produces some output in the process.

Note: Some builders deviate from this general build plan, for example, the builder that checks external links does not need anything more than the parsed doctrees and therefore does not have phases 2–4.

Extension Design

We want the extension to add the following to Sphinx:

- A "todo" directive, containing some content that is marked with "TODO", and only shown in the output if a new config value is set. (Todo entries should not be in the output by default.)
- A "todolist" directive that creates a list of all todo entries throughout the documentation.

For that, we will need to add the following elements to Sphinx:

- New directives, called `todo` and `todolist`.
- New document tree nodes to represent these directives, conventionally also called `todo` and `todolist`. We wouldn't need new nodes if the new directives only produced some content representable by existing nodes.
- A new config value `todo_include_todos` (config value names should start with the extension name, in order to stay unique) that controls whether todo entries make it into the output.
- New event handlers: one for the *`doctree-resolved`* event, to replace the todo and todolist nodes, and one for *`env-purge-doc`* (the reason for that will be covered later).

The Setup Function

The new elements are added in the extension's setup function. Let us create a new Python module called `todo.py` and add the setup function:

[1] http://docutils.sourceforge.net/docs/ref/doctree.html

```
def setup(app):
    app.add_config_value('todo_include_todos', False, 'html')

    app.add_node(todolist)
    app.add_node(todo,
                 html=(visit_todo_node, depart_todo_node),
                 latex=(visit_todo_node, depart_todo_node),
                 text=(visit_todo_node, depart_todo_node))

    app.add_directive('todo', TodoDirective)
    app.add_directive('todolist', TodolistDirective)
    app.connect('doctree-resolved', process_todo_nodes)
    app.connect('env-purge-doc', purge_todos)

    return {'version': '0.1'}   # identifies the version of our extension
```

The calls in this function refer to classes and functions not yet written. What the individual calls do is the following:

- *add_config_value()* lets Sphinx know that it should recognize the new *config value* `todo_include_todos`, whose default value should be `False` (this also tells Sphinx that it is a boolean value).

  If the third argument was `'html'`, HTML documents would be full rebuild if the config value changed its value. This is needed for config values that influence reading (build phase 1).

- *add_node()* adds a new *node class* to the build system. It also can specify visitor functions for each supported output format. These visitor functions are needed when the new nodes stay until phase 4 – since the `todolist` node is always replaced in phase 3, it doesn't need any.

  We need to create the two node classes `todo` and `todolist` later.

- *add_directive()* adds a new *directive*, given by name and class.

  The handler functions are created later.

- Finally, *connect()* adds an *event handler* to the event whose name is given by the first argument. The event handler function is called with several arguments which are documented with the event.

The Node Classes

Let's start with the node classes:

```
from docutils import nodes

class todo(nodes.Admonition, nodes.Element):
    pass

class todolist(nodes.General, nodes.Element):
    pass

def visit_todo_node(self, node):
    self.visit_admonition(node)

def depart_todo_node(self, node):
    self.depart_admonition(node)
```

Node classes usually don't have to do anything except inherit from the standard docutils classes defined in `docutils.nodes`. `todo` inherits from `Admonition` because it should be handled like a note or warning,

`todolist` is just a "general" node.

Note: Many extensions will not have to create their own node classes and work fine with the nodes already provided by docutils[2] and *Sphinx*.

The Directive Classes

A directive class is a class deriving usually from *`docutils.parsers.rst.Directive`*. The directive interface is also covered in detail in the docutils documentation[3]; the important thing is that the class should have attributes that configure the allowed markup, and a `run` method that returns a list of nodes.

The `todolist` directive is quite simple:

```
from docutils.parsers.rst import Directive

class TodolistDirective(Directive):

    def run(self):
        return [todolist('')]
```

An instance of our `todolist` node class is created and returned. The todolist directive has neither content nor arguments that need to be handled.

The `todo` directive function looks like this:

```
from sphinx.util.compat import make_admonition
from sphinx.locale import _

class TodoDirective(Directive):

    # this enables content in the directive
    has_content = True

    def run(self):
        env = self.state.document.settings.env

        targetid = "todo-%d" % env.new_serialno('todo')
        targetnode = nodes.target('', '', ids=[targetid])

        ad = make_admonition(todo, self.name, [_('Todo')], self.options,
                             self.content, self.lineno, self.content_offset,
                             self.block_text, self.state, self.state_machine)

        if not hasattr(env, 'todo_all_todos'):
            env.todo_all_todos = []
        env.todo_all_todos.append({
            'docname': env.docname,
            'lineno': self.lineno,
            'todo': ad[0].deepcopy(),
            'target': targetnode,
        })

        return [targetnode] + ad
```

[2] http://docutils.sourceforge.net/docs/ref/doctree.html
[3] http://docutils.sourceforge.net/docs/ref/rst/directives.html

Several important things are covered here. First, as you can see, you can refer to the build environment instance using `self.state.document.settings.env`.

Then, to act as a link target (from the todolist), the todo directive needs to return a target node in addition to the todo node. The target ID (in HTML, this will be the anchor name) is generated by using `env.new_serialno` which returns a new unique integer on each call and therefore leads to unique target names. The target node is instantiated without any text (the first two arguments).

An admonition is created using a standard docutils function (wrapped in Sphinx for docutils cross-version compatibility). The first argument gives the node class, in our case `todo`. The third argument gives the admonition title (use `arguments` here to let the user specify the title). A list of nodes is returned from `make_admonition`.

Then, the todo node is added to the environment. This is needed to be able to create a list of all todo entries throughout the documentation, in the place where the author puts a `todolist` directive. For this case, the environment attribute `todo_all_todos` is used (again, the name should be unique, so it is prefixed by the extension name). It does not exist when a new environment is created, so the directive must check and create it if necessary. Various information about the todo entry's location are stored along with a copy of the node.

In the last line, the nodes that should be put into the doctree are returned: the target node and the admonition node.

The node structure that the directive returns looks like this:

```
+--------------------+
| target node        |
+--------------------+
+--------------------+
| todo node          |
+--------------------+
  \__+--------------------+
     | admonition title   |
     +--------------------+
     | paragraph          |
     +--------------------+
     | ...                |
     +--------------------+
```

The Event Handlers

Finally, let's look at the event handlers. First, the one for the *env-purge-doc* event:

```
def purge_todos(app, env, docname):
    if not hasattr(env, 'todo_all_todos'):
        return
    env.todo_all_todos = [todo for todo in env.todo_all_todos
                          if todo['docname'] != docname]
```

Since we store information from source files in the environment, which is persistent, it may become out of date when the source file changes. Therefore, before each source file is read, the environment's records of it are cleared, and the *env-purge-doc* event gives extensions a chance to do the same. Here we clear out all todos whose docname matches the given one from the `todo_all_todos` list. If there are todos left in the document, they will be added again during parsing.

The other handler belongs to the *doctree-resolved* event. This event is emitted at the end of phase 3 and allows custom resolving to be done:

```
def process_todo_nodes(app, doctree, fromdocname):
    if not app.config.todo_include_todos:
        for node in doctree.traverse(todo):
            node.parent.remove(node)

    # Replace all todolist nodes with a list of the collected todos.
    # Augment each todo with a backlink to the original location.
    env = app.builder.env

    for node in doctree.traverse(todolist):
        if not app.config.todo_include_todos:
            node.replace_self([])
            continue

        content = []

        for todo_info in env.todo_all_todos:
            para = nodes.paragraph()
            filename = env.doc2path(todo_info['docname'], base=None)
            description = (
                _('(The original entry is located in %s, line %d and can be found ') %
                (filename, todo_info['lineno']))
            para += nodes.Text(description, description)

            # Create a reference
            newnode = nodes.reference('', '')
            innernode = nodes.emphasis(_('here'), _('here'))
            newnode['refdocname'] = todo_info['docname']
            newnode['refuri'] = app.builder.get_relative_uri(
                fromdocname, todo_info['docname'])
            newnode['refuri'] += '#' + todo_info['target']['refid']
            newnode.append(innernode)
            para += newnode
            para += nodes.Text('.)', '.)')

            # Insert into the todolist
            content.append(todo_info['todo'])
            content.append(para)

        node.replace_self(content)
```

It is a bit more involved. If our new "todo_include_todos" config value is false, all todo and todolist nodes are removed from the documents.

If not, todo nodes just stay where and how they are. Todolist nodes are replaced by a list of todo entries, complete with backlinks to the location where they come from. The list items are composed of the nodes from the todo entry and docutils nodes created on the fly: a paragraph for each entry, containing text that gives the location, and a link (reference node containing an italic node) with the backreference. The reference URI is built by `app.builder.get_relative_uri` which creates a suitable URI depending on the used builder, and appending the todo node's (the target's) ID as the anchor name.

15.2.2 Application API

Each Sphinx extension is a Python module with at least a `setup()` function. This function is called at initialization time with one argument, the application object representing the Sphinx process.

class `sphinx.application.`**Sphinx**

This application object has the public API described in the following.

Extension setup

These methods are usually called in an extension's `setup()` function.

Examples of using the Sphinx extension API can be seen in the `sphinx.ext` package.

`Sphinx.`**`setup_extension`**`(`*name*`)`
Load the extension given by the module *name*. Use this if your extension needs the features provided by another extension.

`Sphinx.`**`add_builder`**`(`*builder*`)`
Register a new builder. *builder* must be a class that inherits from *Builder*.

`Sphinx.`**`add_config_value`**`(`*name*, *default*, *rebuild*`)`
Register a configuration value. This is necessary for Sphinx to recognize new values and set default values accordingly. The *name* should be prefixed with the extension name, to avoid clashes. The *default* value can be any Python object. The string value *rebuild* must be one of those values:

- `'env'` if a change in the setting only takes effect when a document is parsed – this means that the whole environment must be rebuilt.
- `'html'` if a change in the setting needs a full rebuild of HTML documents.
- `''` if a change in the setting will not need any special rebuild.

Changed in version 0.4: If the *default* value is a callable, it will be called with the config object as its argument in order to get the default value. This can be used to implement config values whose default depends on other values.

Changed in version 0.6: Changed *rebuild* from a simple boolean (equivalent to `''` or `'env'`) to a string. However, booleans are still accepted and converted internally.

`Sphinx.`**`add_domain`**`(`*domain*`)`
Make the given *domain* (which must be a class; more precisely, a subclass of *Domain*) known to Sphinx.

New in version 1.0.

`Sphinx.`**`override_domain`**`(`*domain*`)`
Make the given *domain* class known to Sphinx, assuming that there is already a domain with its `.name`. The new domain must be a subclass of the existing one.

New in version 1.0.

`Sphinx.`**`add_index_to_domain`**`(`*domain*, *index*`)`
Add a custom *index* class to the domain named *domain*. *index* must be a subclass of *Index*.

New in version 1.0.

`Sphinx.`**`add_event`**`(`*name*`)`
Register an event called *name*. This is needed to be able to emit it.

`Sphinx.`**`set_translator`**`(`*name*, *translator_class*`)`
Register or override a Docutils translator class. This is used to register a custom output translator or to replace a builtin translator. This allows extensions to use custom translator and define custom nodes for the translator (see *add_node()*).

This is a API version of *html_translator_class* for all other builders. Note that if *html_translator_class* is specified and this API is called for html related builders, API overriding takes precedence.

New in version 1.3.

Sphinx.**add_node**(*node*, ***kwds*)

Register a Docutils node class. This is necessary for Docutils internals. It may also be used in the future to validate nodes in the parsed documents.

Node visitor functions for the Sphinx HTML, LaTeX, text and manpage writers can be given as keyword arguments: the keyword should be one or more of `'html'`, `'latex'`, `'text'`, `'man'`, `'texinfo'` or any other supported translators, the value a 2-tuple of `(visit, depart)` methods. `depart` can be `None` if the `visit` function raises `docutils.nodes.SkipNode`. Example:

```
class math(docutils.nodes.Element): pass

def visit_math_html(self, node):
    self.body.append(self.starttag(node, 'math'))
def depart_math_html(self, node):
    self.body.append('</math>')

app.add_node(math, html=(visit_math_html, depart_math_html))
```

Obviously, translators for which you don't specify visitor methods will choke on the node when encountered in a document to translate.

Changed in version 0.5: Added the support for keyword arguments giving visit functions.

Sphinx.**add_directive**(*name*, *func*, *content*, *arguments*, ***options*)

Sphinx.**add_directive**(*name*, *directiveclass*)

Register a Docutils directive. *name* must be the prospective directive name. There are two possible ways to write a directive:

- In the docutils 0.4 style, *obj* is the directive function. *content*, *arguments* and *options* are set as attributes on the function and determine whether the directive has content, arguments and options, respectively. **This style is deprecated.**
- In the docutils 0.5 style, *directiveclass* is the directive class. It must already have attributes named *has_content*, *required_arguments*, *optional_arguments*, *final_argument_whitespace* and *option_spec* that correspond to the options for the function way. See the Docutils docs[4] for details.

  The directive class must inherit from the class `docutils.parsers.rst.Directive`.

For example, the (already existing) *literalinclude* directive would be added like this:

```
from docutils.parsers.rst import directives
add_directive('literalinclude', literalinclude_directive,
              content = 0, arguments = (1, 0, 0),
              linenos = directives.flag,
              language = direcitves.unchanged,
              encoding = directives.encoding)
```

Changed in version 0.6: Docutils 0.5-style directive classes are now supported.

Sphinx.**add_directive_to_domain**(*domain*, *name*, *func*, *content*, *arguments*, ***options*)

Sphinx.**add_directive_to_domain**(*domain*, *name*, *directiveclass*)

Like *add_directive()*, but the directive is added to the domain named *domain*.

New in version 1.0.

Sphinx.**add_role**(*name*, *role*)

Register a Docutils role. *name* must be the role name that occurs in the source, *role* the role function (see the Docutils documentation[5] on details).

[4] http://docutils.sourceforge.net/docs/howto/rst-directives.html
[5] http://docutils.sourceforge.net/docs/howto/rst-roles.html

Sphinx.**add_role_to_domain**(*domain*, *name*, *role*)
Like *add_role()*, but the role is added to the domain named *domain*.

New in version 1.0.

Sphinx.**add_generic_role**(*name*, *nodeclass*)
Register a Docutils role that does nothing but wrap its contents in the node given by *nodeclass*.

New in version 0.6.

Sphinx.**add_object_type**(*directivename*, *rolename*, *indextemplate=''*, *parse_node=None*, *ref_nodeclass=None*, *objname=''*, *doc_field_types=[]*)
This method is a very convenient way to add a new *object* type that can be cross-referenced. It will do this:

- Create a new directive (called *directivename*) for documenting an object. It will automatically add index entries if *indextemplate* is nonempty; if given, it must contain exactly one instance of `%s`. See the example below for how the template will be interpreted.
- Create a new role (called *rolename*) to cross-reference to these object descriptions.
- If you provide *parse_node*, it must be a function that takes a string and a docutils node, and it must populate the node with children parsed from the string. It must then return the name of the item to be used in cross-referencing and index entries. See the `conf.py` file in the source for this documentation for an example.
- The *objname* (if not given, will default to *directivename*) names the type of object. It is used when listing objects, e.g. in search results.

For example, if you have this call in a custom Sphinx extension:

```
app.add_object_type('directive', 'dir', 'pair: %s; directive')
```

you can use this markup in your documents:

```
.. rst:directive:: function

   Document a function.

<...>

See also the :rst:dir:`function` directive.
```

For the directive, an index entry will be generated as if you had prepended

```
.. index:: pair: function; directive
```

The reference node will be of class `literal` (so it will be rendered in a proportional font, as appropriate for code) unless you give the *ref_nodeclass* argument, which must be a docutils node class. Most useful are `docutils.nodes.emphasis` or `docutils.nodes.strong` – you can also use `docutils.nodes.generated` if you want no further text decoration. If the text should be treated as literal (e.g. no smart quote replacement), but not have typewriter styling, use `sphinx.addnodes.literal_emphasis` or `sphinx.addnodes.literal_strong`.

For the role content, you have the same syntactical possibilities as for standard Sphinx roles (see *Cross-referencing syntax*).

This method is also available under the deprecated alias `add_description_unit`.

Sphinx.**add_crossref_type**(*directivename*, *rolename*, *indextemplate=''*, *ref_nodeclass=None*, *objname=''*)
This method is very similar to *add_object_type()* except that the directive it generates must be empty, and will produce no output.

That means that you can add semantic targets to your sources, and refer to them using custom roles instead of generic ones (like *ref*). Example call:

```
app.add_crossref_type('topic', 'topic', 'single: %s', docutils.nodes.emphasis)
```

Example usage:

```
.. topic:: application API

The application API
-------------------

<...>

See also :topic:`this section <application API>`.
```

(Of course, the element following the `topic` directive needn't be a section.)

Sphinx.**add_transform**(*transform*)
Add the standard docutils `Transform` subclass *transform* to the list of transforms that are applied after Sphinx parses a reST document.

Sphinx.**add_javascript**(*filename*)
Add *filename* to the list of JavaScript files that the default HTML template will include. The filename must be relative to the HTML static path, see *the docs for the config value*. A full URI with scheme, like `http://example.org/foo.js`, is also supported.

New in version 0.5.

Sphinx.**add_stylesheet**(*filename*)
Add *filename* to the list of CSS files that the default HTML template will include. Like for *add_javascript()*, the filename must be relative to the HTML static path, or a full URI with scheme.

New in version 1.0.

Sphinx.**add_latex_package**(*packagename*, *options=None*)
Add *packagename* to the list of packages that LaTeX source code will include. If you provide *options*, it will be taken to *usepackage* declaration.

```
app.add_latex_package('mypackage')              # => \usepackage{mypackage}
app.add_latex_package('mypackage', 'foo,bar')   # => \usepackage[foo,bar]{mypackage}
```

New in version 1.3.

Sphinx.**add_lexer**(*alias*, *lexer*)
Use *lexer*, which must be an instance of a Pygments lexer class, to highlight code blocks with the given language *alias*.

New in version 0.6.

Sphinx.**add_autodocumenter**(*cls*)
Add *cls* as a new documenter class for the *sphinx.ext.autodoc* extension. It must be a subclass of `sphinx.ext.autodoc.Documenter`. This allows to auto-document new types of objects. See the source of the autodoc module for examples on how to subclass `Documenter`.

New in version 0.6.

Sphinx.**add_autodoc_attrgetter**(*type*, *getter*)
Add *getter*, which must be a function with an interface compatible to the `getattr()` builtin, as the autodoc attribute getter for objects that are instances of *type*. All cases where autodoc needs to get an attribute of a type are then handled by this function instead of `getattr()`.

New in version 0.6.

Sphinx.**add_search_language**(*cls*)
Add *cls*, which must be a subclass of `sphinx.search.SearchLanguage`, as a support language for building the HTML full-text search index. The class must have a *lang* attribute that indicates the language it should be used for. See *html_search_language*.

New in version 1.1.

Sphinx.**require_sphinx**(*version*)
Compare *version* (which must be a `major.minor` version string, e.g. `'1.1'`) with the version of the running Sphinx, and abort the build when it is too old.

New in version 1.0.

Sphinx.**connect**(*event*, *callback*)
Register *callback* to be called when *event* is emitted. For details on available core events and the arguments of callback functions, please see *Sphinx core events*.

The method returns a "listener ID" that can be used as an argument to *disconnect()*.

Sphinx.**disconnect**(*listener_id*)
Unregister callback *listener_id*.

exception sphinx.application.**ExtensionError**
All these methods raise this exception if something went wrong with the extension API.

Emitting events

Sphinx.**emit**(*event*, **arguments*)
Emit *event* and pass *arguments* to the callback functions. Return the return values of all callbacks as a list. Do not emit core Sphinx events in extensions!

Sphinx.**emit_firstresult**(*event*, **arguments*)
Emit *event* and pass *arguments* to the callback functions. Return the result of the first callback that doesn't return `None`.

New in version 0.5.

Producing messages / logging

The application object also provides support for emitting leveled messages.

Note: There is no "error" call: in Sphinx, errors are defined as things that stop the build; just raise an exception (*sphinx.errors.SphinxError* or a custom subclass) to do that.

Sphinx.**warn**(*message*, *location=None*, *prefix='WARNING: '*)
Emit a warning.

If *location* is given, it should either be a tuple of (docname, lineno) or a string describing the location of the warning as well as possible.

prefix usually should not be changed.

Note: For warnings emitted during parsing, you should use *BuildEnvironment.warn()* since that will collect all warnings during parsing for later output.

Sphinx.**info**(*message=''*, *nonl=False*)
Emit an informational message.

If *nonl* is true, don't emit a newline at the end (which implies that more info output will follow soon.)

Sphinx.**verbose**(*message*, **args*, ***kwargs*)
Emit a verbose informational message.

The message will only be emitted for verbosity levels >= 1 (i.e. at least one `-v` option was given).

The message can contain %-style interpolation placeholders, which is formatted with either the `*args` or `**kwargs` when output.

Sphinx.**debug**(*message*, **args*, ***kwargs*)
Emit a debug-level informational message.

The message will only be emitted for verbosity levels >= 2 (i.e. at least two `-v` options were given).

The message can contain %-style interpolation placeholders, which is formatted with either the `*args` or `**kwargs` when output.

Sphinx.**debug2**(*message*, **args*, ***kwargs*)
Emit a lowlevel debug-level informational message.

The message will only be emitted for verbosity level 3 (i.e. three `-v` options were given).

The message can contain %-style interpolation placeholders, which is formatted with either the `*args` or `**kwargs` when output.

Sphinx core events

These events are known to the core. The arguments shown are given to the registered event handlers. Use *`connect()`* in an extension's `setup` function (note that `conf.py` can also have a `setup` function) to connect handlers to the events. Example:

```
def source_read_handler(app, docname, source):
    print('do something here...')

def setup(app):
    app.connect('source-read', source_read_handler)
```

builder-inited(*app*)
Emitted when the builder object has been created. It is available as `app.builder`.

env-get-outdated(*app*, *env*, *added*, *changed*, *removed*)
Emitted when the environment determines which source files have changed and should be re-read. *added*, *changed* and *removed* are sets of docnames that the environment has determined. You can return a list of docnames to re-read in addition to these.

New in version 1.1.

env-purge-doc(*app*, *env*, *docname*)
Emitted when all traces of a source file should be cleaned from the environment, that is, if the source file is removed or before it is freshly read. This is for extensions that keep their own caches in attributes of the environment.

For example, there is a cache of all modules on the environment. When a source file has been changed, the cache's entries for the file are cleared, since the module declarations could have been removed from the file.

New in version 0.5.

env-before-read-docs (*app*, *env*, *docnames*)
Emitted after the environment has determined the list of all added and changed files and just before it reads them. It allows extension authors to reorder the list of docnames (*inplace*) before processing, or add more docnames that Sphinx did not consider changed (but never add any docnames that are not in `env.found_docs`).

You can also remove document names; do this with caution since it will make Sphinx treat changed files as unchanged.

New in version 1.3.

source-read (*app*, *docname*, *source*)
Emitted when a source file has been read. The *source* argument is a list whose single element is the contents of the source file. You can process the contents and replace this item to implement source-level transformations.

For example, if you want to use `$` signs to delimit inline math, like in LaTeX, you can use a regular expression to replace `$...$` by `:math:`...``.

New in version 0.5.

doctree-read (*app*, *doctree*)
Emitted when a doctree has been parsed and read by the environment, and is about to be pickled. The *doctree* can be modified in-place.

missing-reference (*app*, *env*, *node*, *contnode*)
Emitted when a cross-reference to a Python module or object cannot be resolved. If the event handler can resolve the reference, it should return a new docutils node to be inserted in the document tree in place of the node *node*. Usually this node is a `reference` node containing *contnode* as a child.

Parameters

- **env** – The build environment (`app.builder.env`).
- **node** – The `pending_xref` node to be resolved. Its attributes `reftype`, `reftarget`, `modname` and `classname` attributes determine the type and target of the reference.
- **contnode** – The node that carries the text and formatting inside the future reference and should be a child of the returned reference node.

New in version 0.5.

doctree-resolved (*app*, *doctree*, *docname*)
Emitted when a doctree has been "resolved" by the environment, that is, all references have been resolved and TOCs have been inserted. The *doctree* can be modified in place.

Here is the place to replace custom nodes that don't have visitor methods in the writers, so that they don't cause errors when the writers encounter them.

env-merge-info (*env*, *docnames*, *other*)
This event is only emitted when parallel reading of documents is enabled. It is emitted once for every subprocess that has read some documents.

You must handle this event in an extension that stores data in the environment in a custom location. Otherwise the environment in the main process will not be aware of the information stored in the subprocess.

other is the environment object from the subprocess, *env* is the environment from the main process. *docnames* is a set of document names that have been read in the subprocess.

For a sample of how to deal with this event, look at the standard `sphinx.ext.todo` extension. The implementation is often similar to that of *env-purge-doc*, only that information is not removed, but added to the main environment from the other environment.

New in version 1.3.

env-updated(*app*, *env*)
Emitted when the `update()` method of the build environment has completed, that is, the environment and all doctrees are now up-to-date.

You can return an iterable of docnames from the handler. These documents will then be considered updated, and will be (re-)written during the writing phase.

New in version 0.5.

Changed in version 1.3: The handlers' return value is now used.

html-collect-pages(*app*)
Emitted when the HTML builder is starting to write non-document pages. You can add pages to write by returning an iterable from this event consisting of `(pagename, context, templatename)`.

New in version 1.0.

html-page-context(*app*, *pagename*, *templatename*, *context*, *doctree*)
Emitted when the HTML builder has created a context dictionary to render a template with – this can be used to add custom elements to the context.

The *pagename* argument is the canonical name of the page being rendered, that is, without `.html` suffix and using slashes as path separators. The *templatename* is the name of the template to render, this will be `'page.html'` for all pages from reST documents.

The *context* argument is a dictionary of values that are given to the template engine to render the page and can be modified to include custom values. Keys must be strings.

The *doctree* argument will be a doctree when the page is created from a reST documents; it will be `None` when the page is created from an HTML template alone.

You can return a string from the handler, it will then replace `'page.html'` as the HTML template for this page.

New in version 0.4.

Changed in version 1.3: The return value can now specify a template name.

build-finished(*app*, *exception*)
Emitted when a build has finished, before Sphinx exits, usually used for cleanup. This event is emitted even when the build process raised an exception, given as the *exception* argument. The exception is reraised in the application after the event handlers have run. If the build process raised no exception, *exception* will be `None`. This allows to customize cleanup actions depending on the exception status.

New in version 0.5.

Checking the Sphinx version

Use this to adapt your extension to API changes in Sphinx.

`sphinx.`**version_info**
A tuple of five elements; for Sphinx version 1.2.1 beta 3 this would be `(1, 2, 1, 'beta', 3)`.

New in version 1.2: Before version 1.2, check the string `sphinx.__version__`.

The Config object

class sphinx.config.**Config**
The config object makes the values of all config values available as attributes.

It is available as the `config` attribute on the application and environment objects. For example, to get the value of *language*, use either `app.config.language` or `env.config.language`.

The template bridge

class sphinx.application.**TemplateBridge**
This class defines the interface for a "template bridge", that is, a class that renders templates given a template name and a context.

init(*builder*, *theme=None*, *dirs=None*)
Called by the builder to initialize the template system.

builder is the builder object; you'll probably want to look at the value of `builder.config.templates_path`.

theme is a `sphinx.theming.Theme` object or None; in the latter case, *dirs* can be list of fixed directories to look for templates.

newest_template_mtime()
Called by the builder to determine if output files are outdated because of template changes. Return the mtime of the newest template file that was changed. The default implementation returns `0`.

render(*template*, *context*)
Called by the builder to render a template given as a filename with a specified context (a Python dictionary).

render_string(*template*, *context*)
Called by the builder to render a template given as a string with a specified context (a Python dictionary).

Exceptions

exception sphinx.errors.**SphinxError**
This is the base class for "nice" exceptions. When such an exception is raised, Sphinx will abort the build and present the exception category and message to the user.

Extensions are encouraged to derive from this exception for their custom errors.

Exceptions *not* derived from *SphinxError* are treated as unexpected and shown to the user with a part of the traceback (and the full traceback saved in a temporary file).

category
Description of the exception "category", used in converting the exception to a string ("category: message"). Should be set accordingly in subclasses.

exception sphinx.errors.**ConfigError**
Used for erroneous values or nonsensical combinations of configuration values.

exception sphinx.errors.**ExtensionError**
Used for errors in setting up extensions.

exception sphinx.errors.**ThemeError**
Used for errors to do with themes.

exception sphinx.errors.**VersionRequirementError**
Raised when the docs require a higher Sphinx version than the current one.

15.2.3 Build environment API

class sphinx.environment.**BuildEnvironment**
Attributes

app
Reference to the *Sphinx* (application) object.

config
Reference to the *Config* object.

srcdir
Source directory (the directory containing `conf.py`).

doctreedir
Directory for storing pickled doctrees.

found_docs
A set of all existing docnames.

metadata
Dictionary mapping docnames to "metadata" (see *File-wide metadata*).

titles
Dictionary mapping docnames to the docutils node for their main title.

docname
Returns the docname of the document currently being parsed.

Utility methods

warn(*docname*, *msg*, *lineno=None*)
Emit a warning.

This differs from using `app.warn()` in that the warning may not be emitted instantly, but collected for emitting all warnings after the update of the environment.

warn_node(*msg*, *node*)
Like *warn()*, but with source information taken from *node*.

doc2path(*docname*, *base=True*, *suffix=None*)
Return the filename for the document name.

If *base* is True, return absolute path under self.srcdir. If *base* is None, return relative path to self.srcdir. If *base* is a path string, return absolute path under that. If *suffix* is not None, add it instead of config.source_suffix.

relfn2path(*filename*, *docname=None*)
Return paths to a file referenced from a document, relative to documentation root and absolute.

In the input "filename", absolute filenames are taken as relative to the source dir, while relative filenames are relative to the dir of the containing document.

note_dependency(*filename*)
Add *filename* as a dependency of the current document.

This means that the document will be rebuilt if this file changes.

filename should be absolute or relative to the source directory.

new_serialno(*category=''*)
Return a serial number, e.g. for index entry targets.

The number is guaranteed to be unique in the current document.

note_reread()
Add the current document to the list of documents that will automatically be re-read at the next build.

15.2.4 Builder API

Todo

Expand this.

class sphinx.builders.**Builder**
This is the base class for all builders.

These methods are predefined and will be called from the application:

get_relative_uri(*from_*, *to*, *typ=None*)
Return a relative URI between two source filenames.

May raise environment.NoUri if there's no way to return a sensible URI.

build_all()
Build all source files.

build_specific(*filenames*)
Only rebuild as much as needed for changes in the *filenames*.

build_update()
Only rebuild what was changed or added since last build.

build(*docnames*, *summary=None*, *method='update'*)
Main build method.

First updates the environment, and then calls `write()`.

These methods can be overridden in concrete builder classes:

init()
Load necessary templates and perform initialization. The default implementation does nothing.

get_outdated_docs()
Return an iterable of output files that are outdated, or a string describing what an update build will build.

If the builder does not output individual files corresponding to source files, return a string here. If it does, return an iterable of those files that need to be written.

get_target_uri(*docname*, *typ=None*)
Return the target URI for a document name.

typ can be used to qualify the link characteristic for individual builders.

prepare_writing(*docnames*)
A place where you can add logic before *write_doc()* is run

write_doc(*docname*, *doctree*)
Where you actually write something to the filesystem.

finish()
Finish the building process.

The default implementation does nothing.

15.2.5 Docutils markup API

This section describes the API for adding ReST markup elements (roles and directives).

Roles

Directives

Directives are handled by classes derived from `docutils.parsers.rst.Directive`. They have to be registered by an extension using *Sphinx.add_directive()* or *Sphinx.add_directive_to_domain()*.

class `docutils.parsers.rst.`**Directive**
The markup syntax of the new directive is determined by the follow five class attributes:

required_arguments = 0
Number of required directive arguments.

optional_arguments = 0
Number of optional arguments after the required arguments.

final_argument_whitespace = False
May the final argument contain whitespace?

option_spec = None
Mapping of option names to validator functions.

Option validator functions take a single parameter, the option argument (or `None` if not given), and should validate it or convert it to the proper form. They raise `ValueError` or `TypeError` to indicate failure.

There are several predefined and possibly useful validators in the `docutils.parsers.rst.directives` module.

has_content = False
May the directive have content?

New directives must implement the *run()* method:

run()
This method must process the directive arguments, options and content, and return a list of Docutils/Sphinx nodes that will be inserted into the document tree at the point where the directive was encountered.

Instance attributes that are always set on the directive are:

name
The directive name (useful when registering the same directive class under multiple names).

arguments
The arguments given to the directive, as a list.

options
The options given to the directive, as a dictionary mapping option names to validated/converted values.

content
The directive content, if given, as a `ViewList`.

lineno
The absolute line number on which the directive appeared. This is not always a useful value; use *`srcline`* instead.

src
The source file of the directive.

srcline
The line number in the source file on which the directive appeared.

content_offset
Internal offset of the directive content. Used when calling `nested_parse` (see below).

block_text
The string containing the entire directive.

state
state_machine
The state and state machine which controls the parsing. Used for `nested_parse`.

ViewLists

Docutils represents document source lines in a class `docutils.statemachine.ViewList`. This is a list with extended functionality – for one, slicing creates views of the original list, and also the list contains information about the source line numbers.

The *`Directive.content`* attribute is a ViewList. If you generate content to be parsed as ReST, you have to create a ViewList yourself. Important for content generation are the following points:

- The constructor takes a list of strings (lines) and a source (document) name.
- The `.append()` method takes a line and a source name as well.

Parsing directive content as ReST

Many directives will contain more markup that must be parsed. To do this, use one of the following APIs from the *`Directive.run()`* method:

- `self.state.nested_parse`
- `sphinx.util.nodes.nested_parse_with_titles()` – this allows titles in the parsed content.

Both APIs parse the content into a given node. They are used like this:

```
node = docutils.nodes.paragraph()
# either
nested_parse_with_titles(self.state, self.result, node)
# or
self.state.nested_parse(self.result, 0, node)
```

If you don't need the wrapping node, you can use any concrete node type and return `node.children` from the Directive.

See also:

Creating directives[6] HOWTO of the Docutils documentation

15.2.6 Domain API

class sphinx.domains.**Domain**(*env*)

A Domain is meant to be a group of "object" description directives for objects of a similar nature, and corresponding roles to create references to them. Examples would be Python modules, classes, functions etc., elements of a templating language, Sphinx roles and directives, etc.

Each domain has a separate storage for information about existing objects and how to reference them in *self.data*, which must be a dictionary. It also must implement several functions that expose the object information in a uniform way to parts of Sphinx that allow the user to reference or search for objects in a domain-agnostic way.

About *self.data*: since all object and cross-referencing information is stored on a BuildEnvironment instance, the *domain.data* object is also stored in the *env.domaindata* dict under the key *domain.name*. Before the build process starts, every active domain is instantiated and given the environment object; the *domaindata* dict must then either be nonexistent or a dictionary whose 'version' key is equal to the domain class' *data_version* attribute. Otherwise, *IOError* is raised and the pickled environment is discarded.

clear_doc(*docname*)

Remove traces of a document in the domain-specific inventories.

directive(*name*)

Return a directive adapter class that always gives the registered directive its full name ('domain:name') as self.name.

get_objects()

Return an iterable of "object descriptions", which are tuples with five items:

- *name* – fully qualified name
- *dispname* – name to display when searching/linking
- *type* – object type, a key in self.object_types
- *docname* – the document where it is to be found
- *anchor* – the anchor name for the object
- *priority* – how "important" the object is (determines placement in search results)
  - –1: default priority (placed before full-text matches)
  - –0: object is important (placed before default-priority objects)
  - –2: object is unimportant (placed after full-text matches)
  - –-1: object should not show up in search at all

get_type_name(*type*, *primary=False*)

Return full name for given ObjType.

merge_domaindata(*docnames*, *otherdata*)

Merge in data regarding *docnames* from a different domaindata inventory (coming from a subprocess in parallel builds).

process_doc(*env*, *docname*, *document*)

Process a document after it is read by the environment.

resolve_any_xref(*env*, *fromdocname*, *builder*, *target*, *node*, *contnode*)

Resolve the pending_xref *node* with the given *target*.

The reference comes from an "any" or similar role, which means that we don't know the type. Otherwise, the arguments are the same as for *resolve_xref()*.

The method must return a list (potentially empty) of tuples `('domain:role', newnode)`, where `'domain:role'` is the name of a role that could have created the same reference, e.g. `'py:func'`. `newnode` is what *resolve_xref()* would return.

New in version 1.3.

resolve_xref(*env*, *fromdocname*, *builder*, *typ*, *target*, *node*, *contnode*)
Resolve the pending_xref *node* with the given *typ* and *target*.

This method should return a new node, to replace the xref node, containing the *contnode* which is the markup content of the cross-reference.

If no resolution can be found, None can be returned; the xref node will then given to the 'missing-reference' event, and if that yields no resolution, replaced by *contnode*.

The method can also raise `sphinx.environment.NoUri` to suppress the 'missing-reference' event being emitted.

role(*name*)
Return a role adapter function that always gives the registered role its full name ('domain:name') as the first argument.

dangling_warnings = {}
role name -> a warning message if reference is missing

data_version = 0
data version, bump this when the format of *self.data* changes

directives = {}
directive name -> directive class

indices = []
a list of Index subclasses

initial_data = {}
data value for a fresh environment

label = ''
domain label: longer, more descriptive (used in messages)

name = ''
domain name: should be short, but unique

object_types = {}
type (usually directive) name -> ObjType instance

roles = {}
role name -> role callable

class sphinx.domains.**ObjType**(*lname*, **roles*, ***attrs*)
An ObjType is the description for a type of object that a domain can document. In the object_types attribute of Domain subclasses, object type names are mapped to instances of this class.

Constructor arguments:

- *lname*: localized name of the type (do not include domain name)
- *roles*: all the roles that can refer to an object of this type
- *attrs*: object attributes – currently only "searchprio" is known, which defines the object's priority in the full-text search index, see *Domain.get_objects()*.

class sphinx.domains.**Index**(*domain*)
An Index is the description for a domain-specific index. To add an index to a domain, subclass Index, overriding the three name attributes:

- *name* is an identifier used for generating file names.
- *localname* is the section title for the index.
- *shortname* is a short name for the index, for use in the relation bar in HTML output. Can be empty to disable entries in the relation bar.

and providing a `generate()` method. Then, add the index class to your domain's *indices* list. Extensions can add indices to existing domains using `add_index_to_domain()`.

generate(*docnames=None*)
Return entries for the index given by *name*. If *docnames* is given, restrict to entries referring to these docnames.

The return value is a tuple of `(content, collapse)`, where *collapse* is a boolean that determines if sub-entries should start collapsed (for output formats that support collapsing sub-entries).

content is a sequence of `(letter, entries)` tuples, where *letter* is the "heading" for the given *entries*, usually the starting letter.

entries is a sequence of single entries, where a single entry is a sequence `[name, subtype, docname, anchor, extra, qualifier, descr]`. The items in this sequence have the following meaning:

- *name* – the name of the index entry to be displayed
- *subtype* – sub-entry related type: 0 – normal entry 1 – entry with sub-entries 2 – sub-entry
- *docname* – docname where the entry is located
- *anchor* – anchor for the entry within *docname*
- *extra* – extra info for the entry
- *qualifier* – qualifier for the description
- *descr* – description for the entry

Qualifier and description are not rendered e.g. in LaTeX output.

15.2.7 Doctree node classes added by Sphinx

Nodes for domain-specific object descriptions

class sphinx.addnodes.**desc**(*rawsource=''*, **children*, ***attributes*)
Node for object descriptions.

This node is similar to a "definition list" with one definition. It contains one or more `desc_signature` and a `desc_content`.

class sphinx.addnodes.**desc_signature**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for object signatures.

The "term" part of the custom Sphinx definition list.

class sphinx.addnodes.**desc_addname**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for additional name parts (module name, class name).

class sphinx.addnodes.**desc_type**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for return types or object type names.

class sphinx.addnodes.**desc_returns**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for a "returns" annotation (a la -> in Python).

class sphinx.addnodes.**desc_name**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for the main object name.

class sphinx.addnodes.**desc_parameterlist**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for a general parameter list.

class sphinx.addnodes.**desc_parameter**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for a single parameter.

class sphinx.addnodes.**desc_optional**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for marking optional parts of the parameter list.

class sphinx.addnodes.**desc_annotation**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for signature annotations (not Python 3-style annotations).

class sphinx.addnodes.**desc_content**(*rawsource=''*, **children*, ***attributes*)
Node for object description content.

This is the "definition" part of the custom Sphinx definition list.

New admonition-like constructs

class sphinx.addnodes.**versionmodified**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for version change entries.

Currently used for "versionadded", "versionchanged" and "deprecated" directives.

class sphinx.addnodes.**seealso**(*rawsource=''*, **children*, ***attributes*)
Custom "see also" admonition.

Other paragraph-level nodes

class sphinx.addnodes.**compact_paragraph**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for a compact paragraph (which never makes a <p> node).

New inline nodes

class sphinx.addnodes.**index**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for index entries.

This node is created by the `index` directive and has one attribute, `entries`. Its value is a list of 4-tuples of `(entrytype, entryname, target, ignored)`.

entrytype is one of "single", "pair", "double", "triple".

class sphinx.addnodes.**pending_xref**(*rawsource=''*, **children*, ***attributes*)
Node for cross-references that cannot be resolved without complete information about all documents.

These nodes are resolved before writing output, in BuildEnvironment.resolve_references.

class sphinx.addnodes.**literal_emphasis**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node that behaves like *emphasis*, but further text processors are not applied (e.g. smartypants for HTML output).

class sphinx.addnodes.**abbreviation**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for abbreviations with explanations.

class sphinx.addnodes.**download_reference**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for download references, similar to pending_xref.

Special nodes

class sphinx.addnodes.**only**(*rawsource=''*, **children*, ***attributes*)
Node for "only" directives (conditional inclusion based on tags).

class sphinx.addnodes.**meta**(*rawsource=''*, **children*, ***attributes*)
Node for meta directive – same as docutils' standard meta node, but pickleable.

class sphinx.addnodes.**highlightlang**(*rawsource=''*, **children*, ***attributes*)
Inserted to set the highlight language and line number options for subsequent code blocks.

You should not need to generate the nodes below in extensions.

class sphinx.addnodes.**glossary**(*rawsource=''*, **children*, ***attributes*)
Node to insert a glossary.

class sphinx.addnodes.**toctree**(*rawsource=''*, **children*, ***attributes*)
Node for inserting a "TOC tree".

class sphinx.addnodes.**start_of_file**(*rawsource=''*, **children*, ***attributes*)
Node to mark start of a new file, used in the LaTeX builder only.

class sphinx.addnodes.**productionlist**(*rawsource=''*, **children*, ***attributes*)
Node for grammar production lists.

Contains `production` nodes.

class sphinx.addnodes.**production**(*rawsource=''*, *text=''*, **children*, ***attributes*)
Node for a single grammar production rule.

class sphinx.addnodes.**termsep**(*rawsource=''*, **children*, ***attributes*)
Separates two terms within a <term> node.

CHAPTER 16

Sphinx Web Support

New in version 1.1.

Sphinx provides a Python API to easily integrate Sphinx documentation into your web application. To learn more read the *Web Support Quick Start*.

16.1 Web Support Quick Start

16.1.1 Building Documentation Data

To make use of the web support package in your application you'll need to build the data it uses. This data includes pickle files representing documents, search indices, and node data that is used to track where comments and other things are in a document. To do this you will need to create an instance of the *WebSupport* class and call its *build()* method:

```
from sphinx.websupport import WebSupport

support = WebSupport(srcdir='/path/to/rst/sources/',
                     builddir='/path/to/build/outdir',
                     search='xapian')

support.build()
```

This will read reStructuredText sources from *srcdir* and place the necessary data in *builddir*. The *builddir* will contain two sub-directories: one named "data" that contains all the data needed to display documents, search through documents, and add comments to documents. The other directory will be called "static" and contains static files that should be served from "/static".

Note: If you wish to serve static files from a path other than "/static", you can do so by providing the *staticdir* keyword argument when creating the *WebSupport* object.

16.1.2 Integrating Sphinx Documents Into Your Webapp

Now that the data is built, it's time to do something useful with it. Start off by creating a *WebSupport* object for your application:

```
from sphinx.websupport import WebSupport
```

```
support = WebSupport(datadir='/path/to/the/data',
                     search='xapian')
```

You'll only need one of these for each set of documentation you will be working with. You can then call its *get_document()* method to access individual documents:

```
contents = support.get_document('contents')
```

This will return a dictionary containing the following items:

- **body**: The main body of the document as HTML
- **sidebar**: The sidebar of the document as HTML
- **relbar**: A div containing links to related documents
- **title**: The title of the document
- **css**: Links to CSS files used by Sphinx
- **script**: JavaScript containing comment options

This dict can then be used as context for templates. The goal is to be easy to integrate with your existing templating system. An example using Jinja2[1] is:

```
{%- extends "layout.html" %}

{%- block title %}
    {{ document.title }}
{%- endblock %}

{% block css %}
    {{ super() }}
    {{ document.css|safe }}
    <link rel="stylesheet" href="/static/websupport-custom.css" type="text/css">
{% endblock %}

{%- block script %}
    {{ super() }}
    {{ document.script|safe }}
{%- endblock %}

{%- block relbar %}
    {{ document.relbar|safe }}
{%- endblock %}

{%- block body %}
    {{ document.body|safe }}
{%- endblock %}

{%- block sidebar %}
    {{ document.sidebar|safe }}
{%- endblock %}
```

Authentication

To use certain features such as voting, it must be possible to authenticate users. The details of the authentication are left to your application. Once a user has been authenticated you can pass the user's details

[1] http://jinja.pocoo.org/

to certain *WebSupport* methods using the *username* and *moderator* keyword arguments. The web support package will store the username with comments and votes. The only caveat is that if you allow users to change their username you must update the websupport package's data:

```
support.update_username(old_username, new_username)
```

username should be a unique string which identifies a user, and *moderator* should be a boolean representing whether the user has moderation privileges. The default value for *moderator* is `False`.

An example Flask[2] function that checks whether a user is logged in and then retrieves a document is:

```
from sphinx.websupport.errors import *

@app.route('/<path:docname>')
def doc(docname):
    username = g.user.name if g.user else ''
    moderator = g.user.moderator if g.user else False
    try:
        document = support.get_document(docname, username, moderator)
    except DocumentNotFoundError:
        abort(404)
    return render_template('doc.html', document=document)
```

The first thing to notice is that the *docname* is just the request path. This makes accessing the correct document easy from a single view. If the user is authenticated, then the username and moderation status are passed along with the docname to *get_document()*. The web support package will then add this data to the `COMMENT_OPTIONS` that are used in the template.

Note: This only works works if your documentation is served from your document root. If it is served from another directory, you will need to prefix the url route with that directory, and give the *docroot* keyword argument when creating the web support object:

```
support = WebSupport(..., docroot='docs')

@app.route('/docs/<path:docname>')
```

16.1.3 Performing Searches

To use the search form built-in to the Sphinx sidebar, create a function to handle requests to the url 'search' relative to the documentation root. The user's search query will be in the GET parameters, with the key *q*. Then use the *get_search_results()* method to retrieve search results. In Flask[3] that would be like this:

```
@app.route('/search')
def search():
    q = request.args.get('q')
    document = support.get_search_results(q)
    return render_template('doc.html', document=document)
```

Note that we used the same template to render our search results as we did to render our documents. That's because *get_search_results()* returns a context dict in the same format that *get_document()* does.

[2] http://flask.pocoo.org/
[3] http://flask.pocoo.org/

16.1.4 Comments & Proposals

Now that this is done it's time to define the functions that handle the AJAX calls from the script. You will need three functions. The first function is used to add a new comment, and will call the web support method *add_comment()*:

```
@app.route('/docs/add_comment', methods=['POST'])
def add_comment():
    parent_id = request.form.get('parent', '')
    node_id = request.form.get('node', '')
    text = request.form.get('text', '')
    proposal = request.form.get('proposal', '')
    username = g.user.name if g.user is not None else 'Anonymous'
    comment = support.add_comment(text, node_id='node_id',
                                  parent_id='parent_id',
                                  username=username, proposal=proposal)
    return jsonify(comment=comment)
```

You'll notice that both a *parent_id* and *node_id* are sent with the request. If the comment is being attached directly to a node, *parent_id* will be empty. If the comment is a child of another comment, then *node_id* will be empty. Then next function handles the retrieval of comments for a specific node, and is aptly named *get_data()*:

```
@app.route('/docs/get_comments')
def get_comments():
    username = g.user.name if g.user else None
    moderator = g.user.moderator if g.user else False
    node_id = request.args.get('node', '')
    data = support.get_data(node_id, username, moderator)
    return jsonify(**data)
```

The final function that is needed will call *process_vote()*, and will handle user votes on comments:

```
@app.route('/docs/process_vote', methods=['POST'])
def process_vote():
    if g.user is None:
        abort(401)
    comment_id = request.form.get('comment_id')
    value = request.form.get('value')
    if value is None or comment_id is None:
        abort(400)
    support.process_vote(comment_id, g.user.id, value)
    return "success"
```

16.1.5 Comment Moderation

By default, all comments added through *add_comment()* are automatically displayed. If you wish to have some form of moderation, you can pass the *displayed* keyword argument:

```
comment = support.add_comment(text, node_id='node_id',
                              parent_id='parent_id',
                              username=username, proposal=proposal,
                              displayed=False)
```

You can then create a new view to handle the moderation of comments. It will be called when a moderator decides a comment should be accepted and displayed:

```
@app.route('/docs/accept_comment', methods=['POST'])
def accept_comment():
    moderator = g.user.moderator if g.user else False
    comment_id = request.form.get('id')
    support.accept_comment(comment_id, moderator=moderator)
    return 'OK'
```

Rejecting comments happens via comment deletion.

To perform a custom action (such as emailing a moderator) when a new comment is added but not displayed, you can pass callable to the *WebSupport* class when instantiating your support object:

```
def moderation_callback(comment):
    """Do something..."""

support = WebSupport(..., moderation_callback=moderation_callback)
```

The moderation callback must take one argument, which will be the same comment dict that is returned by `add_comment()`.

16.2 The WebSupport Class

class sphinx.websupport.**WebSupport**

The main API class for the web support package. All interactions with the web support package should occur through this class.

The class takes the following keyword arguments:

srcdir The directory containing reStructuredText source files.

builddir The directory that build data and static files should be placed in. This should be used when creating a *WebSupport* object that will be used to build data.

datadir The directory that the web support data is in. This should be used when creating a *WebSupport* object that will be used to retrieve data.

search This may contain either a string (e.g. 'xapian') referencing a built-in search adapter to use, or an instance of a subclass of *BaseSearch*.

storage This may contain either a string representing a database uri, or an instance of a subclass of *StorageBackend*. If this is not provided, a new sqlite database will be created.

moderation_callback A callable to be called when a new comment is added that is not displayed. It must accept one argument: a dictionary representing the comment that was added.

staticdir If static files are served from a location besides `'/static'`, this should be a string with the name of that location (e.g. `'/static_files'`).

docroot If the documentation is not served from the base path of a URL, this should be a string specifying that path (e.g. `'docs'`).

16.2.1 Methods

WebSupport.**build**()

Build the documentation. Places the data into the *outdir* directory. Use it like this:

```
    support = WebSupport(srcdir, builddir, search='xapian')
    support.build()
```

This will read reStructured text files from *srcdir*. Then it will build the pickles and search index, placing them into *builddir*. It will also save node data to the database.

WebSupport.**get_document**(*docname*, *username=''*, *moderator=False*)

Load and return a document from a pickle. The document will be a dict object which can be used to render a template:

```
support = WebSupport(datadir=datadir)
support.get_document('index', username, moderator)
```

In most cases *docname* will be taken from the request path and passed directly to this function. In Flask, that would be something like this:

```
@app.route('/<path:docname>')
def index(docname):
    username = g.user.name if g.user else ''
    moderator = g.user.moderator if g.user else False
    try:
        document = support.get_document(docname, username,
                                        moderator)
    except DocumentNotFoundError:
        abort(404)
    render_template('doc.html', document=document)
```

The document dict that is returned contains the following items to be used during template rendering.

- **body**: The main body of the document as HTML
- **sidebar**: The sidebar of the document as HTML
- **relbar**: A div containing links to related documents
- **title**: The title of the document
- **css**: Links to css files used by Sphinx
- **script**: Javascript containing comment options

This raises `DocumentNotFoundError` if a document matching *docname* is not found.

Parameters **docname** – the name of the document to load.

WebSupport.**get_data**(*node_id*, *username=None*, *moderator=False*)

Get the comments and source associated with *node_id*. If *username* is given vote information will be included with the returned comments. The default CommentBackend returns a dict with two keys, *source*, and *comments*. *source* is raw source of the node and is used as the starting point for proposals a user can add. *comments* is a list of dicts that represent a comment, each having the following items:

Key	Contents
text	The comment text.
username	The username that was stored with the comment.
id	The comment's unique identifier.
rating	The comment's current rating.
age	The time in seconds since the comment was added.
time	A dict containing time information. It contains the following keys: year, month, day, hour, minute, second, iso, and delta. *iso* is the time formatted in ISO 8601 format. *delta* is a printable form of how old the comment is (e.g. "3 hours ago").
vote	If *user_id* was given, this will be an integer representing the vote. 1 for an upvote, -1 for a downvote, or 0 if unvoted.
node	The id of the node that the comment is attached to. If the comment's parent is another comment rather than a node, this will be null.
parent	The id of the comment that this comment is attached to if it is not attached to a node.
children	A list of all children, in this format.
proposal_diff	An HTML representation of the differences between the the current source and the user's proposed source.

Parameters

- **node_id** – the id of the node to get comments for.
- **username** – the username of the user viewing the comments.
- **moderator** – whether the user is a moderator.

WebSupport.**add_comment**(*text*, *node_id=''*, *parent_id=''*, *displayed=True*, *username=None*, *time=None*, *proposal=None*, *moderator=False*)

Add a comment to a node or another comment. Returns the comment in the same format as `get_comments()`. If the comment is being attached to a node, pass in the node's id (as a string) with the node keyword argument:

```
comment = support.add_comment(text, node_id=node_id)
```

If the comment is the child of another comment, provide the parent's id (as a string) with the parent keyword argument:

```
comment = support.add_comment(text, parent_id=parent_id)
```

If you would like to store a username with the comment, pass in the optional *username* keyword argument:

```
comment = support.add_comment(text, node=node_id,
                              username=username)
```

Parameters

- **parent_id** – the prefixed id of the comment's parent.
- **text** – the text of the comment.
- **displayed** – for moderation purposes
- **username** – the username of the user making the comment.
- **time** – the time the comment was created, defaults to now.

WebSupport.**process_vote**(*comment_id*, *username*, *value*)
Process a user's vote. The web support package relies on the API user to perform authentication. The API user will typically receive a comment_id and value from a form, and then make sure the user is authenticated. A unique username must be passed in, which will also be used to retrieve the user's past voting data. An example, once again in Flask:

```
@app.route('/docs/process_vote', methods=['POST'])
def process_vote():
    if g.user is None:
        abort(401)
    comment_id = request.form.get('comment_id')
    value = request.form.get('value')
    if value is None or comment_id is None:
        abort(400)
    support.process_vote(comment_id, g.user.name, value)
    return "success"
```

Parameters

- **comment_id** – the comment being voted on
- **username** – the unique username of the user voting
- **value** – 1 for an upvote, -1 for a downvote, 0 for an unvote.

WebSupport.**get_search_results**(*q*)
Perform a search for the query *q*, and create a set of search results. Then render the search results as html and return a context dict like the one created by *get_document()*:

```
document = support.get_search_results(q)
```

Parameters **q** – the search query

16.3 Search Adapters

To create a custom search adapter you will need to subclass the *BaseSearch* class. Then create an instance of the new class and pass that as the *search* keyword argument when you create the *WebSupport* object:

```
support = WebSupport(srcdir=srcdir,
                     builddir=builddir,
                     search=MySearch())
```

For more information about creating a custom search adapter, please see the documentation of the *BaseSearch* class below.

class sphinx.websupport.search.**BaseSearch**
Defines an interface for search adapters.

16.3.1 BaseSearch Methods

The following methods are defined in the BaseSearch class. Some methods do not need to be overridden, but some (*add_document()* and *handle_query()*) must be overridden in your subclass. For a working example, look at the built-in adapter for whoosh.

BaseSearch.**init_indexing**(*changed=[]*)
Called by the builder to initialize the search indexer. *changed* is a list of pagenames that will be reindexed. You may want to remove these from the search index before indexing begins.

Parameters **changed** – a list of pagenames that will be re-indexed

BaseSearch.**finish_indexing**()
Called by the builder when writing has been completed. Use this to perform any finalization or cleanup actions after indexing is complete.

BaseSearch.**feed**(*pagename*, *title*, *doctree*)
Called by the builder to add a doctree to the index. Converts the *doctree* to text and passes it to *add_document()*. You probably won't want to override this unless you need access to the *doctree*. Override *add_document()* instead.

Parameters

- **pagename** – the name of the page to be indexed
- **title** – the title of the page to be indexed
- **doctree** – is the docutils doctree representation of the page

BaseSearch.**add_document**(*pagename*, *title*, *text*)
Called by *feed()* to add a document to the search index. This method should should do everything necessary to add a single document to the search index.

pagename is name of the page being indexed. It is the combination of the source files relative path and filename, minus the extension. For example, if the source file is "ext/builders.rst", the *pagename* would be "ext/builders". This will need to be returned with search results when processing a query.

Parameters

- **pagename** – the name of the page being indexed
- **title** – the page's title
- **text** – the full text of the page

BaseSearch.**query**(*q*)
Called by the web support api to get search results. This method compiles the regular expression to be used when *extracting context*, then calls *handle_query()*. You won't want to override this unless you don't want to use the included *extract_context()* method. Override *handle_query()* instead.

Parameters **q** – the search query string.

BaseSearch.**handle_query**(*q*)
Called by *query()* to retrieve search results for a search query *q*. This should return an iterable containing tuples of the following format:

```
(<path>, <title>, <context>)
```

path and *title* are the same values that were passed to *add_document()*, and *context* should be a short text snippet of the text surrounding the search query in the document.

The *extract_context()* method is provided as a simple way to create the *context*.

Parameters **q** – the search query

BaseSearch.**extract_context**(*text*, *length=240*)
Extract the context for the search query from the document's full *text*.

Parameters

- **text** – the full text of the document to create the context for
- **length** – the length of the context snippet to return.

16.4 Storage Backends

To create a custom storage backend you will need to subclass the *StorageBackend* class. Then create an instance of the new class and pass that as the *storage* keyword argument when you create the *WebSupport* object:

```
support = WebSupport(srcdir=srcdir,
                     builddir=builddir,
                     storage=MyStorage())
```

For more information about creating a custom storage backend, please see the documentation of the *StorageBackend* class below.

class sphinx.websupport.storage.**StorageBackend**
Defines an interface for storage backends.

16.4.1 StorageBackend Methods

StorageBackend.**pre_build**()
Called immediately before the build process begins. Use this to prepare the StorageBackend for the addition of nodes.

StorageBackend.**add_node**(*id*, *document*, *source*)
Add a node to the StorageBackend.

Parameters

- **id** – a unique id for the comment.
- **document** – the name of the document the node belongs to.
- **source** – the source files name.

StorageBackend.**post_build**()
Called after a build has completed. Use this to finalize the addition of nodes if needed.

StorageBackend.**add_comment**(*text*, *displayed*, *username*, *time*, *proposal*, *node_id*, *parent_id*, *moderator*)
Called when a comment is being added.

Parameters

- **text** – the text of the comment
- **displayed** – whether the comment should be displayed
- **username** – the name of the user adding the comment
- **time** – a date object with the time the comment was added
- **proposal** – the text of the proposal the user made
- **node_id** – the id of the node that the comment is being added to
- **parent_id** – the id of the comment's parent comment.
- **moderator** – whether the user adding the comment is a moderator

StorageBackend.**delete_comment**(*comment_id*, *username*, *moderator*)
Delete a comment.

Raises UserNotAuthorizedError if moderator is False and *username* doesn't match the username on the comment.

Parameters

- **comment_id** – The id of the comment being deleted.
- **username** – The username of the user requesting the deletion.
- **moderator** – Whether the user is a moderator.

StorageBackend.**get_data**(*node_id*, *username*, *moderator*)
Called to retrieve all data for a node. This should return a dict with two keys, *source* and *comments* as described by *WebSupport*'s *get_data()* method.

Parameters

- **node_id** – The id of the node to get data for.
- **username** – The name of the user requesting the data.
- **moderator** – Whether the requestor is a moderator.

StorageBackend.**process_vote**(*comment_id*, *username*, *value*)
Process a vote that is being cast. *value* will be either -1, 0, or 1.

Parameters

- **comment_id** – The id of the comment being voted on.
- **username** – The username of the user casting the vote.
- **value** – The value of the vote being cast.

StorageBackend.**update_username**(*old_username*, *new_username*)
If a user is allowed to change their username this method should be called so that there is not stagnate data in the storage system.

Parameters

- **old_username** – The username being changed.
- **new_username** – What the username is being changed to.

StorageBackend.**accept_comment**(*comment_id*)
Called when a moderator accepts a comment. After the method is called the comment should be displayed to all users.

Parameters **comment_id** – The id of the comment being accepted.

CHAPTER 17

Sphinx FAQ

This is a list of Frequently Asked Questions about Sphinx. Feel free to suggest new entries!

17.1 How do I...

... create PDF files without LaTeX? You can use rst2pdf[1] version 0.12 or greater which comes with built-in Sphinx integration. See the *Available builders* section for details.

... get section numbers? They are automatic in LaTeX output; for HTML, give a `:numbered:` option to the `toctree` directive where you want to start numbering.

... customize the look of the built HTML files? Use themes, see HTML theming support.

... add global substitutions or includes? Add them in the `rst_epilog` config value.

... display the whole TOC tree in the sidebar? Use the `toctree` callable in a custom layout template, probably in the `sidebartoc` block.

... write my own extension? See the *extension tutorial*.

... convert from my existing docs using MoinMoin markup? The easiest way is to convert to xhtml, then convert xhtml to reST[2]. You'll still need to mark up classes and such, but the headings and code examples come through cleanly.

... create HTML slides from Sphinx documents? See the "Hieroglyph" package at https://github.com/nyergler/hieroglyph.

For many more extensions and other contributed stuff, see the sphinx-contrib[3] repository.

17.2 Using Sphinx with...

Read the Docs https://readthedocs.org is a documentation hosting service based around Sphinx. They will host sphinx documentation, along with supporting a number of other features including version support, PDF generation, and more. The Getting Started[4] guide is a good place to start.

Epydoc There's a third-party extension providing an api role[5] which refers to Epydoc's API docs for a given identifier.

[1] http://rst2pdf.googlecode.com
[2] http://docutils.sourceforge.net/sandbox/xhtml2rest/xhtml2rest.py
[3] https://bitbucket.org/birkenfeld/sphinx-contrib/
[4] http://read-the-docs.readthedocs.org/en/latest/getting_started.html
[5] http://git.savannah.gnu.org/cgit/kenozooid.git/tree/doc/extapi.py

Doxygen Michael Jones is developing a reST/Sphinx bridge to doxygen called breathe[6].

SCons Glenn Hutchings has written a SCons build script to build Sphinx documentation; it is hosted here: https://bitbucket.org/zondo/sphinx-scons

PyPI Jannis Leidel wrote a setuptools command[7] that automatically uploads Sphinx documentation to the PyPI package documentation area at http://pythonhosted.org/.

GitHub Pages Directories starting with underscores are ignored by default which breaks static files in Sphinx. GitHub's preprocessor can be disabled[8] to support Sphinx HTML output properly.

MediaWiki See https://bitbucket.org/kevindunn/sphinx-wiki, a project by Kevin Dunn.

Google Analytics You can use a custom `layout.html` template, like this:

```
{% extends "!layout.html" %}

{%- block extrahead %}
{{ super() }}
<script type="text/javascript">
  var _gaq = _gaq || [];
  _gaq.push(['_setAccount', 'XXX account number XXX']);
  _gaq.push(['_trackPageview']);
</script>
{% endblock %}

{% block footer %}
{{ super() }}
<div class="footer">This page uses <a href="http://analytics.google.com/">
Google Analytics</a> to collect statistics. You can disable it by blocking
the JavaScript coming from www.google-analytics.com.
<script type="text/javascript">
  (function() {
    var ga = document.createElement('script');
    ga.src = ('https:' == document.location.protocol ?
              'https://ssl' : 'http://www') + '.google-analytics.com/ga.js';
    ga.setAttribute('async', 'true');
    document.documentElement.firstChild.appendChild(ga);
  })();
</script>
</div>
{% endblock %}
```

17.3 Epub info

The following list gives some hints for the creation of epub files:

- Split the text into several files. The longer the individual HTML files are, the longer it takes the ebook reader to render them. In extreme cases, the rendering can take up to one minute.
- Try to minimize the markup. This also pays in rendering time.
- For some readers you can use embedded or external fonts using the CSS `@font-face` directive. This is *extremely* useful for code listings which are often cut at the right margin. The default Courier font (or variant) is quite wide and you can only display up to 60 characters on a line. If you replace it

[6] https://github.com/michaeljones/breathe/tree/master
[7] https://pypi.python.org/pypi/Sphinx-PyPI-upload
[8] https://github.com/blog/572-bypassing-jekyll-on-github-pages

with a narrower font, you can get more characters on a line. You may even use FontForge[9] and create narrow variants of some free font. In my case I get up to 70 characters on a line.

You may have to experiment a little until you get reasonable results.

- Test the created epubs. You can use several alternatives. The ones I am aware of are Epubcheck[10], Calibre[11], FBreader[12] (although it does not render the CSS), and Bookworm[13]. For bookworm you can download the source from http://code.google.com/p/threepress/ and run your own local server.
- Large floating divs are not displayed properly. If they cover more than one page, the div is only shown on the first page. In that case you can copy the `epub.css` from the `sphinx/themes/epub/static/` directory to your local `_static/` directory and remove the float settings.
- Files that are inserted outside of the `toctree` directive must be manually included. This sometimes applies to appendixes, e.g. the glossary or the indices. You can add them with the *`epub_post_files`* option.
- The handling of the epub cover page differs from the reStructuredText procedure which automatically resolves image paths and puts the images into the `_images` directory. For the epub cover page put the image in the *`html_static_path`* directory and reference it with its full path in the *`epub_cover`* config option.

17.4 Texinfo info

There are two main programs for reading Info files, `info` and GNU Emacs. The `info` program has less features but is available in most Unix environments and can be quickly accessed from the terminal. Emacs provides better font and color display and supports extensive customization (of course).

17.4.1 Displaying Links

One noticeable problem you may encounter with the generated Info files is how references are displayed. If you read the source of an Info file, a reference to this section would look like:

```
* note Displaying Links: target-id
```

In the stand-alone reader, `info`, references are displayed just as they appear in the source. Emacs, on the other-hand, will by default replace `*note:` with `see` and hide the `target-id`. For example:

> *Displaying Links*

The exact behavior of how Emacs displays references is dependent on the variable `Info-hide-note-references`. If set to the value of `hide`, Emacs will hide both the `*note:` part and the `target-id`. This is generally the best way to view Sphinx-based documents since they often make frequent use of links and do not take this limitation into account. However, changing this variable affects how all Info documents are displayed and most due take this behavior into account.

If you want Emacs to display Info files produced by Sphinx using the value `hide` for `Info-hide-note-references` and the default value for all other Info files, try adding the following Emacs Lisp code to your start-up file, `~/.emacs.d/init.el`.

[9] http://fontforge.org/
[10] http://code.google.com/p/epubcheck/
[11] http://calibre-ebook.com/
[12] http://fbreader.org/
[13] http://oreilly.com/bookworm/index.html

```
(defadvice info-insert-file-contents (after
                                      sphinx-info-insert-file-contents
                                      activate)
  "Hack to make `Info-hide-note-references' buffer-local and
automatically set to `hide' iff it can be determined that this file
was created from a Texinfo file generated by Docutils or Sphinx."
  (set (make-local-variable 'Info-hide-note-references)
       (default-value 'Info-hide-note-references))
  (save-excursion
    (save-restriction
      (widen) (goto-char (point-min))
      (when (re-search-forward
             "^Generated by \\(Sphinx\\|Docutils\\)"
             (save-excursion (search-forward "\x1f" nil t)) t)
        (set (make-local-variable 'Info-hide-note-references)
             'hide)))))
```

17.4.2 Notes

The following notes may be helpful if you want to create Texinfo files:

- Each section corresponds to a different `node` in the Info file.
- Colons (`:`) cannot be properly escaped in menu entries and xrefs. They will be replaced with semicolons (`;`).
- Links to external Info files can be created using the somewhat official URI scheme `info`. For example:

```
info:Texinfo#makeinfo_options
```

  which produces:

  info:Texinfo#makeinfo_options

- Inline markup

  The standard formatting for `*strong*` and `_emphasis_` can result in ambiguous output when used to markup parameter names and other values. Since this is a fairly common practice, the default formatting has been changed so that `emphasis` and `strong` are now displayed like `'literal'`s.

  The standard formatting can be re-enabled by adding the following to your `conf.py`:

```
texinfo_elements = {'preamble': """
@definfoenclose strong,*,*
@definfoenclose emph,_,_
"""}
```

CHAPTER 18

Glossary

builder A class (inheriting from *Builder*) that takes parsed documents and performs an action on them. Normally, builders translate the documents to an output format, but it is also possible to use the builder builders that e.g. check for broken links in the documentation, or build coverage information.

See *Available builders* for an overview over Sphinx's built-in builders.

configuration directory The directory containing `conf.py`. By default, this is the same as the *source directory*, but can be set differently with the **-c** command-line option.

directive A reStructuredText markup element that allows marking a block of content with special meaning. Directives are supplied not only by docutils, but Sphinx and custom extensions can add their own. The basic directive syntax looks like this:

```
.. directivename:: argument ...
   :option: value

   Content of the directive.
```

See *Directives* for more information.

document name Since reST source files can have different extensions (some people like `.txt`, some like `.rst` – the extension can be configured with *source_suffix*) and different OSes have different path separators, Sphinx abstracts them: *document names* are always relative to the *source directory*, the extension is stripped, and path separators are converted to slashes. All values, parameters and such referring to "documents" expect such document names.

Examples for document names are `index`, `library/zipfile`, or `reference/datamodel/types`. Note that there is no leading or trailing slash.

domain A domain is a collection of markup (reStructuredText *directive*s and *role*s) to describe and link to *object*s belonging together, e.g. elements of a programming language. Directive and role names in a domain have names like `domain:name`, e.g. `py:function`.

Having domains means that there are no naming problems when one set of documentation wants to refer to e.g. C++ and Python classes. It also means that extensions that support the documentation of whole new languages are much easier to write. For more information about domains, see the chapter *Sphinx Domains*.

environment A structure where information about all documents under the root is saved, and used for cross-referencing. The environment is pickled after the parsing stage, so that successive runs only need to read and parse new and changed documents.

master document The document that contains the root *toctree* directive.

object The basic building block of Sphinx documentation. Every "object directive" (e.g. `function` or *`object`*) creates such a block; and most objects can be cross-referenced to.

role A reStructuredText markup element that allows marking a piece of text. Like directives, roles are extensible. The basic syntax looks like this: `:rolename:`content``. See *Inline markup* for details.

source directory The directory which, including its subdirectories, contains all source files for one Sphinx project.

CHAPTER 19

Sphinx Developer's Guide

Abstract

This document describes the development process of Sphinx, a documentation system used by developers to document systems used by other developers to develop other systems that may also be documented using Sphinx.

The Sphinx source code is managed using Git and is hosted on Github.

```
git clone git://github.com/sphinx-doc/sphinx
```

Community

sphinx-users <sphinx-users@googlegroups.com[1]> Mailing list for user support.

sphinx-dev <sphinx-dev@googlegroups.com[2]> Mailing list for development related discussions.

#sphinx-doc on irc.freenode.net IRC channel for development questions and user support.

19.1 Bug Reports and Feature Requests

If you have encountered a problem with Sphinx or have an idea for a new feature, please submit it to the issue tracker[3] on Github or discuss it on the sphinx-dev mailing list.

For bug reports, please include the output produced during the build process and also the log file Sphinx creates after it encounters an un-handled exception. The location of this file should be shown towards the end of the error message.

Including or providing a link to the source files involved may help us fix the issue. If possible, try to create a minimal project that produces the error and post that instead.

19.2 Contributing to Sphinx

The recommended way for new contributors to submit code to Sphinx is to fork the repository on Github and then submit a pull request after committing the changes. The pull request will then need to be approved by one of the core developers before it is merged into the main repository.

[3] https://github.com/sphinx-doc/sphinx/issues

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug.
2. If you feel uncomfortable or uncertain about an issue or your changes, feel free to email sphinx-dev@googlegroups.com[4].
3. Fork the repository[5] on Github to start making your changes to the **master** branch for next major version, or **stable** branch for next minor version.
4. Write a test which shows that the bug was fixed or that the feature works as expected.
5. Send a pull request and bug the maintainer until it gets merged and published. Make sure to add yourself to AUTHORS[6] and the change to CHANGES[7].

19.2.1 Getting Started

These are the basic steps needed to start developing on Sphinx.

1. Create an account on Github.
2. Fork the main Sphinx repository (sphinx-doc/sphinx[8]) using the Github interface.
3. Clone the forked repository to your machine.

```
git clone https://github.com/USERNAME/sphinx
cd sphinx
```

4. Checkout the appropriate branch.

   For changes that should be included in the next minor release (namely bug fixes), use the `stable` branch.

```
git checkout stable
```

   For new features or other substantial changes that should wait until the next major release, use the `master` branch.

5. Optional: setup a virtual environment.

```
virtualenv ~/sphinxenv
. ~/sphinxenv/bin/activate
pip install -e .
```

6. Create a new working branch. Choose any name you like.

```
git checkout -b feature-xyz
```

7. Hack, hack, hack.

   For tips on working with the code, see the *Coding Guide*.

8. Test, test, test. Possible steps:

   - Run the unit tests:

```
pip install nose mock
make test
```

[4] sphinx-dev@googlegroups.com
[5] https://github.com/sphinx-doc/sphinx
[6] https://github.com/sphinx-doc/sphinx/blob/master/AUTHORS
[7] https://github.com/sphinx-doc/sphinx/blob/master/CHANGES
[8] https://github.com/sphinx-doc/sphinx

- Build the documentation and check the output for different builders:

```
cd doc
make clean html latexpdf
```

- Run the unit tests under different Python environments using **tox**:

```
pip install tox
tox -v
```

- Add a new unit test in the `tests` directory if you can.
- For bug fixes, first add a test that fails without your changes and passes after they are applied.
- Tests that need a sphinx-build run should be integrated in one of the existing test modules if possible. New tests that to `@with_app` and then `build_all` for a few assertions are not good since *the test suite should not take more than a minute to run*.

9. Please add a bullet point to `CHANGES` if the fix or feature is not trivial (small doc updates, typo fixes). Then commit:

```
git commit -m '#42: Add useful new feature that does this.'
```

   Github recognizes certain phrases that can be used to automatically update the issue tracker.

   For example:

```
git commit -m 'Closes #42: Fix invalid markup in docstring of Foo.bar.'
```

   would close issue #42.

10. Push changes in the branch to your forked repository on Github.

```
git push origin feature-xyz
```

11. Submit a pull request from your branch to the respective branch (`master` or `stable`) on `sphinx-doc/sphinx` using the Github interface.
12. Wait for a core developer to review your changes.

19.2.2 Core Developers

The core developers of Sphinx have write access to the main repository. They can commit changes, accept/reject pull requests, and manage items on the issue tracker.

You do not need to be a core developer or have write access to be involved in the development of Sphinx. You can submit patches or create pull requests from forked repositories and have a core developer add the changes for you.

The following are some general guidelines for core developers:

- Questionable or extensive changes should be submitted as a pull request instead of being committed directly to the main repository. The pull request should be reviewed by another core developer before it is merged.
- Trivial changes can be committed directly but be sure to keep the repository in a good working state and that all tests pass before pushing your changes.
- When committing code written by someone else, please attribute the original author in the commit message and any relevant `CHANGES` entry.

19.2.3 Locale updates

The parts of messages in Sphinx that go into builds are translated into several locales. The translations are kept as gettext `.po` files translated from the master template `sphinx/locale/sphinx.pot`.

Sphinx uses Babel[9] to extract messages and maintain the catalog files. It is integrated in `setup.py`:

- Use `python setup.py extract_messages` to update the `.pot` template.
- Use `python setup.py update_catalog` to update all existing language catalogs in `sphinx/locale/*/LC_MESSAGES` with the current messages in the template file.
- Use `python setup.py compile_catalog` to compile the `.po` files to binary `.mo` files and `.js` files.

When an updated `.po` file is submitted, run compile_catalog to commit both the source and the compiled catalogs.

When a new locale is submitted, add a new directory with the ISO 639-1 language identifier and put `sphinx.po` in there. Don't forget to update the possible values for *`language`* in `doc/config.rst`.

The Sphinx core messages can also be translated on Transifex[10]. There exists a client tool named `tx` in the Python package "transifex_client", which can be used to pull translations in `.po` format from Transifex. To do this, go to `sphinx/locale` and then run `tx pull -f -l LANG` where LANG is an existing language identifier. It is good practice to run `python setup.py update_catalog` afterwards to make sure the `.po` file has the canonical Babel formatting.

19.3 Coding Guide

- Try to use the same code style as used in the rest of the project. See the Pocoo Styleguide[11] for more information.
- For non-trivial changes, please update the `CHANGES` file. If your changes alter existing behavior, please document this.
- New features should be documented. Include examples and use cases where appropriate. If possible, include a sample that is displayed in the generated output.
- When adding a new configuration variable, be sure to document it and update `sphinx/quickstart.py` if it's important enough.
- Use the included **`utils/check_sources.py`** script to check for common formatting issues (trailing whitespace, lengthy lines, etc).
- Add appropriate unit tests.

19.3.1 Debugging Tips

- Delete the build cache before building documents if you make changes in the code by running the command `make clean` or using the *`sphinx-build -E`* option.
- Use the *`sphinx-build -P`* option to run Pdb on exceptions.
- Use `node.pformat()` and `node.asdom().toxml()` to generate a printable representation of the document structure.

[9] http://babel.edgewall.org
[10] https://www.transifex.com/
[11] http://flask.pocoo.org/docs/styleguide/

- Set the configuration variable *keep_warnings* to True so warnings will be displayed in the generated output.
- Set the configuration variable *nitpicky* to True so that Sphinx will complain about references without a known target.
- Set the debugging options in the Docutils configuration file[12].
- JavaScript stemming algorithms in *sphinx/search/*.py* (except *en.py*) are generated by this modified snowballcode generator[13]. Generated JSX[14] files are in this repository[15]. You can get the resulting JavaScript files using the following command:

```
$ npm install
$ node_modules/.bin/grunt build # -> dest/*.global.js
```

[12] http://docutils.sourceforge.net/docs/user/config.html
[13] https://github.com/shibukawa/snowball
[14] http://jsx.github.io/
[15] https://github.com/shibukawa/snowball-stemmer.jsx

CHAPTER 20

Changes in Sphinx

20.1 Release 1.4 (in development)

20.1.1 Incompatible changes

20.1.2 Features added

- #1962: when adding directives, roles or nodes from an extension, warn if such an element is already present (built-in or added by another extension).
- #1935: Make "numfig_format" overridable in latex_elements.
- #1909: Add "doc" references to Intersphinx inventories.
- C++ type alias support (e.g., `.. type:: T = int`)
- C++ template support for classes, functions, type aliases, and variables (#1729, #1314).
- C++, added new scope management directives `namespace-push` and `namespace-pop`.

20.1.3 Bugs fixed

- #1976: Avoid "2.0" version of Babel because it doesn't work with Windows environment.
- Add a "default.css" stylesheet (which imports "classic.css") for compatibility.
- #1788: graphviz extension raises exception when caption option is present.
- #1789: `:pyobject:` option of `literalinclude` directive includes following lines after class definitions.
- #1790: `literalinclude` strips empty lines at the head and tail.
- #1913: C++, fix assert bug for enumerators in next-to-global and global scope.
- #1802: load plugin themes automatically when theme.conf use it as 'inherit'. Thanks to Takayuki Hirai.
- #1794: custom theme extended from alabaster or sphinx_rtd_theme can't find base theme.
- #1834: compatibility for docutils-0.13: handle_io_errors keyword argument for docutils.io.FileInput cause TypeError.
- #1823: '.' as <module_path> for sphinx-apidoc cause an unfriendly error. Now '.' is converted to absolute path automatically.

- Fix a crash when setting up extensions which do not support metadata.
- #1784: Provide non-minified JS code in `sphinx/search/*.py`
- #1822, #1892: Fix regression for #1061. autosummary can't generate doc for imported members since sphinx-1.3b3. Thanks to Eric Larson.
- #1793, #1819: "see also" misses a linebreak in text output. Thanks to Takayuki Hirai.
- #1780, #1866: "make text" shows "class" keyword twice. Thanks to Takayuki Hirai.
- #1871: Fix for LaTeX output of tables with one column and multirows.
- Work around the lack of the HTMLParserError exception in Python 3.5.
- #1949: Use `safe_getattr` in the coverage builder to avoid aborting with descriptors that have custom behavior.
- #1915: Do not generate smart quotes in doc field type annotations.
- #1796: On py3, automated .mo building caused UnicodeDecodeError.
- #1923: Use babel features only if the babel latex element is nonempty.
- #1942: Fix a KeyError in websupport.
- #1903: Fix strange id generation for glossary terms.
- #1796, On py3, automated .mo building cause UnicodeDecodeError
- `make text` will crush if a definition list item has more than 1 classifiers as: `term : classifier1 : classifier2`.
- #1855: make gettext generates broken po file for definition lists with classifier.
- #1869: Fix problems when dealing with files containing non-ASCII characters. Thanks to Marvin Schmidt.
- #1798: Fix building LaTeX with references in titles.
- #1725: On py2 environment, doctest with using non-ASCII characters causes `'ascii' codec can't decode byte` exception.
- #1540: Fix RuntimeError with circular referenced toctree
- #1983: i18n translation feature breaks references which uses section name.
- #1990: Use caption of toctree to title of tableofcontents in LaTeX
- #1987: Fix ampersand is ignored in `:menuselection:` and `:guilabel:` on LaTeX builder
- #1994: More supporting non-standard parser (like recommonmark parser) for Translation and Web-Support feature. Now node.rawsource is fall backed to node.astext() during docutils transforming.
- C++, fix parsing of 'signed char' and 'unsigned char' as types.
- C++, add missing support for 'friend' functions.

20.1.4 Documentation

20.2 Release 1.3.1 (released Mar 17, 2015)

20.2.1 Bugs fixed

- #1769: allows generating quickstart files/dirs for destination dir that doesn't overwrite existent files/dirs. Thanks to WAKAYAMA shirou.
- #1773: sphinx-quickstart doesn't accept non-ASCII character as a option argument.
- #1766: the message "least Python 2.6 to run" is at best misleading.
- #1772: cross reference in docstrings like `:param .write:` breaks building.
- #1770, #1774: `literalinclude` with empty file occurs exception. Thanks to Takayuki Hirai.
- #1777: Sphinx 1.3 can't load extra theme. Thanks to tell-k.
- #1776: `source_suffix = ['.rst']` cause unfriendly error on prior version.
- #1771: automated .mo building doesn't work properly.
- #1783: Autodoc: Python2 Allow unicode string in __all__. Thanks to Jens Hedegaard Nielsen.
- #1781: Setting *`html_domain_indices`* to a list raises a type check warnings.

20.3 Release 1.3 (released Mar 10, 2015)

20.3.1 Incompatible changes

- Roles `ref`, `term` and `menusel` now don't generate emphasis[1] nodes anymore. If you want to keep italic style, adapt your stylesheet.
- Role `numref` uses `%s` as special character to indicate position of figure numbers instead `#` symbol.

20.3.2 Features added

- Add convenience directives and roles to the C++ domain: directive `cpp:var` as alias for `cpp:member`, role `:cpp:var` as alias for `:cpp:member`, and role *`any`* for cross-reference to any C++ declaraction. #1577, #1744
- The *`source_suffix`* config value can now be a list of multiple suffixes.
- Add the ability to specify source parsers by source suffix with the *`source_parsers`* config value.
- #1675: A new builder, AppleHelpBuilder, has been added that builds Apple Help Books.

20.3.3 Bugs fixed

- 1.3b3 change breaks a previous gettext output that contains duplicated msgid such as "foo bar" and "version changes in 1.3: foo bar".

[1] http://docutils.sourceforge.net/docs/ref/rst/roles.html#emphasis

- #1745: latex builder cause maximum recursion depth exceeded when a footnote has a footnote mark itself.
- #1748: SyntaxError in sphinx/ext/ifconfig.py with Python 2.6.
- #1658, #1750: No link created (and warning given) if option does not begin with -, / or +. Thanks to Takayuki Hirai.
- #1753: C++, added missing support for more complex declarations.
- #1700: Add `:caption:` option for *`toctree`*.
- #1742: `:name:` option is provided for *`toctree`*, *`code-block`* and *`literalinclude`* dirctives.
- #1756: Incorrect section titles in search that was introduced from 1.3b3.
- #1746: C++, fixed name lookup procedure, and added missing lookups in declarations.
- #1765: C++, fix old id generation to use fully qualified names.

20.3.4 Documentation

- #1651: Add `vartype` field descritpion for python domain.

20.4 Release 1.3b3 (released Feb 24, 2015)

20.4.1 Incompatible changes

- Dependency requirement updates: docutils 0.11, Pygments 2.0
- The `gettext_enables` config value has been renamed to *`gettext_additional_targets`*.
- #1735: Use https://doc.python.org/ instead of `http` protocol. It was used for *`sphinx.ext.intersphinx`* and some documentation.

20.4.2 Features added

- #1346: Add new default theme;
  - Add 'alabaster' theme.
  - Add 'sphinx_rtd_theme' theme.
  - The 'default' html theme has been renamed to 'classic'. 'default' is still available, however it will emit notice a recommendation that using new 'alabaster' theme.
- Added `highlight_options` configuration value.
- The `language` config value is now available in the HTML templates.
- The `env-updated` event can now return a value, which is interpreted as an iterable of additional docnames that need to be rewritten.
- #772: Support for scoped and unscoped enums in C++. Enumerators in unscoped enums are injected into the parent scope in addition to the enum scope.
- Add `todo_include_todos` config option to quickstart conf file, handled as described in documentation.
- HTML breadcrumb items tag has class "nav-item" and "nav-item-N" (like nav-item-0, 1, 2...).

- New option *sphinx-quickstart --use-make-mode* for generating Makefile that use sphinx-build make-mode.
- #1235: i18n: several node can be translated if it is set to *gettext_additional_targets* in conf.py. Supported nodes are:
  - 'literal-block'
  - 'doctest-block'
  - 'raw'
  - 'image'
- #1227: Add *html_scaled_image_link* config option to conf.py, to control scaled image link.

20.4.3 Bugs fixed

- LaTeX writer now generates correct markup for cells spanning multiple rows.
- #1674: Do not crash if a module's `__all__` is not a list of strings.
- #1629: Use VerbatimBorderColor to add frame to code-block in LaTeX
- On windows, make-mode didn't work on Win32 platform if sphinx was invoked as `python sphinx-build.py`.
- #1687: linkcheck now treats 401 Unauthorized responses as "working".
- #1690: toctrees with `glob` option now can also contain entries for single documents with explicit title.
- #1591: html search results for C++ elements now has correct interpage links.
- bizstyle theme: nested long title pages make long breadcrumb that breaks page layout.
- bizstyle theme: all breadcrumb items become 'Top' on some mobile browser (iPhone5s safari).
- #1722: restore `toctree()` template function behavior that was changed at 1.3b1.
- #1732: i18n: localized table caption raises exception.
- #1718: `:numref:` does not work with capital letters in the label
- #1630: resolve CSS conflicts, `div.container` css target for literal block wrapper now renamed to `div.literal-block-wrapper`.
- `sphinx.util.pycompat` has been restored in its backwards-compatibility; slated for removal in Sphinx 1.4.
- #1719: LaTeX writer does not respect `numref_format` option in captions

20.5 Release 1.3b2 (released Dec 5, 2014)

20.5.1 Incompatible changes

- update bundled ez_setup.py for setuptools-7.0 that requires Python 2.6 or later.

20.5.2 Features added

- #1597: Added possibility to return a new template name from *html-page-context*.
- PR#314, #1150: Configuration values are now checked for their type. A warning is raised if the configured and the default value do not have the same type and do not share a common non-trivial base class.

20.5.3 Bugs fixed

- PR#311: sphinx-quickstart does not work on python 3.4.
- Fix *autodoc_docstring_signature* not working with signatures in class docstrings.
- Rebuilding cause crash unexpectedly when source files were added.
- #1607: Fix a crash when building latexpdf with "howto" class
- #1251: Fix again. Sections which depth are lower than :tocdepth: should not be shown on localtoc sidebar.
- make-mode didn't work on Win32 platform if sphinx was installed by wheel package.

20.6 Release 1.3b1 (released Oct 10, 2014)

20.6.1 Incompatible changes

- Dropped support for Python 2.5, 3.1 and 3.2.
- Dropped support for docutils versions up to 0.9.
- Removed the `sphinx.ext.oldcmarkup` extension.
- The deprecated config values `exclude_trees`, `exclude_dirnames` and `unused_docs` have been removed.
- A new node, `sphinx.addnodes.literal_strong`, has been added, for text that should appear literally (i.e. no smart quotes) in strong font. Custom writers will have to be adapted to handle this node.
- PR#269, #1476: replace `<tt>` tag by `<code>`. User customized stylesheets should be updated If the css contain some styles for `tt>` tag. Thanks to Takeshi Komiya.
- #1543: *templates_path* is automatically added to *exclude_patterns* to avoid reading autosummary rst templates in the templates directory.
- Custom domains should implement the new *Domain.resolve_any_xref* method to make the *any* role work properly.
- gettext builder: gettext doesn't emit uuid information to generated pot files by default. Please set `True` to *gettext_uuid* to emit uuid information. Additionally, if the `python-levenshtein` 3rd-party package is installed, it will improve the calculation time.
- gettext builder: disable extracting/apply 'index' node by default. Please set 'index' to `gettext_enables` to enable extracting index entries.
- PR#307: Add frame to code-block in LaTeX. Thanks to Takeshi Komiya.

20.6.2 Features added

- Add support for Python 3.4.
- Add support for docutils 0.12
- Added `sphinx.ext.napoleon` extension for NumPy and Google style docstring support.
- Added support for parallel reading (parsing) of source files with the *sphinx-build -j* option. Third-party extensions will need to be checked for compatibility and may need to be adapted if they store information in the build environment object. See *env-merge-info*.
- Added the *any* role that can be used to find a cross-reference of *any* type in *any* domain. Custom domains should implement the new *Domain.resolve_any_xref* method to make this work properly.
- Exception logs now contain the last 10 messages emitted by Sphinx.
- Added support for extension versions (a string returned by `setup()`, these can be shown in the traceback log files). Version requirements for extensions can be specified in projects using the new *needs_extensions* config value.
- Changing the default role within a document with the default-role[2] directive is now supported.
- PR#214: Added stemming support for 14 languages, so that the built-in document search can now handle these. Thanks to Shibukawa Yoshiki.
- PR#296, PR#303, #76: numfig feature: Assign numbers to figures, tables and code-blocks. This feature is configured with *numfig*, *numfig_secnum_depth* and *numfig_format*. Also *numref* role is available. Thanks to Takeshi Komiya.
- PR#202: Allow "." and "~" prefixed references in `:param:` doc fields for Python.
- PR#184: Add *autodoc_mock_imports*, allowing to mock imports of external modules that need not be present when autodocumenting.
- #925: Allow list-typed config values to be provided on the command line, like `-D key=val1,val2`.
- #668: Allow line numbering of *code-block* and *literalinclude* directives to start at an arbitrary line number, with a new `lineno-start` option.
- PR#172, PR#266: The *code-block* and *literalinclude* directives now can have a `caption` option that shows a filename before the code in the output. Thanks to Nasimul Haque, Takeshi Komiya.
- Prompt for the document language in sphinx-quickstart.
- PR#217: Added config values to suppress UUID and location information in generated gettext catalogs.
- PR#236, #1456: apidoc: Add a -M option to put module documentation before submodule documentation. Thanks to Wes Turner and Luc Saffre.
- #1434: Provide non-minified JS files for jquery.js and underscore.js to clarify the source of the minified files.
- PR#252, #1291: Windows color console support. Thanks to meu31.
- PR#255: When generating latex references, also insert latex target/anchor for the ids defined on the node. Thanks to Olivier Heurtier.
- PR#229: Allow registration of other translators. Thanks to Russell Sim.

[2] http://docutils.sourceforge.net/docs/ref/rst/directives.html#default-role

- Add app.set_translator() API to register or override a Docutils translator class like *html_translator_class*.
- PR#267, #1134: add 'diff' parameter to literalinclude. Thanks to Richard Wall and WAKAYAMA shirou.
- PR#272: Added 'bizstyle' theme. Thanks to Shoji KUMAGAI.
- Automatically compile `*.mo` files from `*.po` files when *gettext_auto_build* is True (default) and `*.po` is newer than `*.mo` file.
- #623: *sphinx.ext.viewcode* supports imported function/class aliases.
- PR#275: *sphinx.ext.intersphinx* supports multiple target for the inventory. Thanks to Brigitta Sipocz.
- PR#261: Added the *env-before-read-docs* event that can be connected to modify the order of documents before they are read by the environment.
- #1284: Program options documented with *option* can now start with `+`.
- PR#291: The caption of *code-block* is recognised as a title of ref target. Thanks to Takeshi Komiya.
- PR#298: Add new API: *add_latex_package()*. Thanks to Takeshi Komiya.
- #1344: add `gettext_enables` to enable extracting 'index' to gettext catalog output / applying translation catalog to generated documentation.
- PR#301, #1583: Allow the line numbering of the directive *literalinclude* to match that of the included file, using a new `lineno-match` option. Thanks to Jeppe Pihl.
- PR#299: add various options to sphinx-quickstart. Quiet mode option `--quiet` will skips wizard mode. Thanks to WAKAYAMA shirou.
- #1623: Return types specified with `:rtype:` are now turned into links if possible.

20.6.3 Bugs fixed

- #1438: Updated jQuery version from 1.8.3 to 1.11.1.
- #1568: Fix a crash when a "centered" directive contains a reference.
- Now sphinx.ext.autodoc works with python-2.5 again.
- #1563: *add_search_language()* raises AssertionError for correct type of argument. Thanks to rikoman.
- #1174: Fix smart quotes being applied inside roles like *program* or *makevar*.
- PR#235: comment db schema of websupport lacked a length of the node_id field. Thanks to solos.
- #1466,PR#241: Fix failure of the cpp domain parser to parse C+11 "variadic templates" declarations. Thanks to Victor Zverovich.
- #1459,PR#244: Fix default mathjax js path point to `http://` that cause mixed-content error on HTTPS server. Thanks to sbrandtb and robo9k.
- PR#157: autodoc remove spurious signatures from @property decorated attributes. Thanks to David Ham.
- PR#159: Add coverage targets to quickstart generated Makefile and make.bat. Thanks to Matthias Troffaes.
- #1251: When specifying toctree :numbered: option and :tocdepth: metadata, sub section number that is larger depth than `:tocdepth:` is shrunk.

- PR#260: Encode underscore in citation labels for latex export. Thanks to Lennart Fricke.
- PR#264: Fix could not resolve xref for figure node with :name: option. Thanks to Takeshi Komiya.
- PR#265: Fix could not capture caption of graphviz node by xref. Thanks to Takeshi Komiya.
- PR#263, #1013, #1103: Rewrite of C++ domain. Thanks to Jakob Lykke Andersen.
  - Hyperlinks to all found nested names and template arguments (#1103).
  - Support for function types everywhere, e.g., in std::function<bool(int, int)> (#1013).
  - Support for virtual functions.
  - Changed interpretation of function arguments to following standard prototype declarations, i.e., void f(arg) means that arg is the type of the argument, instead of it being the name.
  - Updated tests.
  - Updated documentation with elaborate description of what declarations are supported and how the namespace declarations influence declaration and cross-reference lookup.
  - Index names may be different now. Elements are indexed by their fully qualified name. It should be rather easy to change this behaviour and potentially index by namespaces/classes as well.
- PR#258, #939: Add dedent option for *code-block* and *literalinclude*. Thanks to Zafar Siddiqui.
- PR#268: Fix numbering section does not work at singlehtml mode. It still ad-hoc fix because there is a issue that section IDs are conflicted. Thanks to Takeshi Komiya.
- PR#273, #1536: Fix RuntimeError with numbered circular toctree. Thanks to Takeshi Komiya.
- PR#274: Set its URL as a default title value if URL appears in toctree. Thanks to Takeshi Komiya.
- PR#276, #1381: *rfc* and *pep* roles support custom link text. Thanks to Takeshi Komiya.
- PR#277, #1513: highlights for function pointers in argument list of *c:function*. Thanks to Takeshi Komiya.
- PR#278: Fix section entries were shown twice if toctree has been put under only directive. Thanks to Takeshi Komiya.
- #1547: pgen2 tokenizer doesn't recognize `...` literal (Ellipsis for py3).
- PR#294: On LaTeX builder, wrap float environment on writing literal_block to avoid separation of caption and body. Thanks to Takeshi Komiya.
- PR#295, #1520: `make.bat latexpdf` mechanism to `cd` back to the current directory. Thanks to Peter Suter.
- PR#297, #1571: Add imgpath property to all builders. It make easier to develop builder extensions. Thanks to Takeshi Komiya.
- #1584: Point to master doc in HTML "top" link.
- #1585: Autosummary of modules broken in Sphinx-1.2.3.
- #1610: Sphinx cause AttributeError when MeCab search option is enabled and python-mecab is not installed.
- #1674: Do not crash if a module's `__all__` is not a list of strings.
- #1673: Fix crashes with *nitpick_ignore* and `:doc:` references.
- #1686: ifconfig directive doesn't care about default config values.
- #1642: Fix only one search result appearing in Chrome.

20.6.4 Documentation

- Add clarification about the syntax of tags. (`doc/markup/misc.rst`)

20.7 Release 1.2.3 (released Sep 1, 2014)

20.7.1 Features added

- #1518: `sphinx-apidoc` command now has a `--version` option to show version information and exit
- New locales: Hebrew, European Portuguese, Vietnamese.

20.7.2 Bugs fixed

- #636: Keep straight single quotes in literal blocks in the LaTeX build.
- #1419: Generated i18n sphinx.js files are missing message catalog entries from '.js_t' and '.html'. The issue was introduced from Sphinx-1.1
- #1363: Fix i18n: missing python domain's cross-references with currentmodule directive or currentclass directive.
- #1444: autosummary does not create the description from attributes docstring.
- #1457: In python3 environment, make linkcheck cause "Can't convert 'bytes' object to str implicitly" error when link target url has a hash part. Thanks to Jorge_C.
- #1467: Exception on Python3 if nonexistent method is specified by automethod
- #1441: autosummary can't handle nested classes correctly.
- #1499: With non-callable `setup` in a conf.py, now sphinx-build emits a user-friendly error message.
- #1502: In autodoc, fix display of parameter defaults containing backslashes.
- #1226: autodoc, autosummary: importing setup.py by automodule will invoke setup process and execute `sys.exit()`. Now sphinx avoids SystemExit exception and emits warnings without unexpected termination.
- #1503: py:function directive generate incorrectly signature when specifying a default parameter with an empty list `[]`. Thanks to Geert Jansen.
- #1508: Non-ASCII filename raise exception on make singlehtml, latex, man, texinfo and changes.
- #1531: On Python3 environment, docutils.conf with 'source_link=true' in the general section cause type error.
- PR#270, #1533: Non-ASCII docstring cause UnicodeDecodeError when uses with inheritance-diagram directive. Thanks to WAKAYAMA shirou.
- PR#281, PR#282, #1509: TODO extension not compatible with websupport. Thanks to Takeshi Komiya.
- #1477: gettext does not extract nodes.line in a table or list.
- #1544: `make text` generates wrong table when it has empty table cells.
- #1522: Footnotes from table get displayed twice in LaTeX. This problem has been appeared from Sphinx-1.2.1 by #949.

- #508: Sphinx every time exit with zero when is invoked from setup.py command. ex. `python setup.py build_sphinx -b doctest` return zero even if doctest failed.

20.8 Release 1.2.2 (released Mar 2, 2014)

20.8.1 Bugs fixed

- PR#211: When checking for existence of the *`html_logo`* file, check the full relative path and not the basename.
- PR#212: Fix traceback with autodoc and `__init__` methods without docstring.
- PR#213: Fix a missing import in the setup command.
- #1357: Option names documented by *`option`* are now again allowed to not start with a dash or slash, and referencing them will work correctly.
- #1358: Fix handling of image paths outside of the source directory when using the "wildcard" style reference.
- #1374: Fix for autosummary generating overly-long summaries if first line doesn't end with a period.
- #1383: Fix Python 2.5 compatibility of sphinx-apidoc.
- #1391: Actually prevent using "pngmath" and "mathjax" extensions at the same time in sphinx-quickstart.
- #1386: Fix bug preventing more than one theme being added by the entry point mechanism.
- #1370: Ignore "toctree" nodes in text writer, instead of raising.
- #1364: Fix 'make gettext' fails when the '.. todolist::' directive is present.
- #1367: Fix a change of PR#96 that break sphinx.util.docfields.Field.make_field interface/behavior for `item` argument usage.

20.8.2 Documentation

- Extended the *documentation about building extensions*.

20.9 Release 1.2.1 (released Jan 19, 2014)

20.9.1 Bugs fixed

- #1335: Fix autosummary template overloading with exclamation prefix like `{% extends "!autosummary/class.rst" %}` cause infinite recursive function call. This was caused by PR#181.
- #1337: Fix autodoc with `autoclass_content="both"` uses useless `object.__init__` docstring when class does not have `__init__`. This was caused by a change for #1138.
- #1340: Can't search alphabetical words on the HTML quick search generated with language='ja'.
- #1319: Do not crash if the *`html_logo`* file does not exist.
- #603: Do not use the HTML-ized title for building the search index (that resulted in "literal" being found on every page with a literal in the title).

- #751: Allow production lists longer than a page in LaTeX by using longtable.
- #764: Always look for stopwords lowercased in JS search.
- #814: autodoc: Guard against strange type objects that don't have `__bases__`.
- #932: autodoc: Do not crash if `__doc__` is not a string.
- #933: Do not crash if an *option* value is malformed (contains spaces but no option name).
- #908: On Python 3, handle error messages from LaTeX correctly in the pngmath extension.
- #943: In autosummary, recognize "first sentences" to pull from the docstring if they contain uppercase letters.
- #923: Take the entire LaTeX document into account when caching pngmath-generated images. This rebuilds them correctly when *pngmath_latex_preamble* changes.
- #901: Emit a warning when using docutils' new "math" markup without a Sphinx math extension active.
- #845: In code blocks, when the selected lexer fails, display line numbers nevertheless if configured.
- #929: Support parsed-literal blocks in LaTeX output correctly.
- #949: Update the tabulary.sty packed with Sphinx.
- #1050: Add anonymous labels into `objects.inv` to be referenced via *intersphinx*.
- #1095: Fix print-media stylesheet being included always in the "scrolls" theme.
- #1085: Fix current classname not getting set if class description has `:noindex:` set.
- #1181: Report option errors in autodoc directives more gracefully.
- #1155: Fix autodocumenting C-defined methods as attributes in Python 3.
- #1233: Allow finding both Python classes and exceptions with the "class" and "exc" roles in intersphinx.
- #1198: Allow "image" for the "figwidth" option of the figure[3] directive as documented by docutils.
- #1152: Fix pycode parsing errors of Python 3 code by including two grammar versions for Python 2 and 3, and loading the appropriate version for the running Python version.
- #1017: Be helpful and tell the user when the argument to *option* does not match the required format.
- #1345: Fix two bugs with *nitpick_ignore*; now you don't have to remove the store environment for changes to have effect.
- #1072: In the JS search, fix issues searching for upper-cased words by lowercasing words before stemming.
- #1299: Make behavior of the *math* directive more consistent and avoid producing empty environments in LaTeX output.
- #1308: Strip HTML tags from the content of "raw" nodes before feeding it to the search indexer.
- #1249: Fix duplicate LaTeX page numbering for manual documents.
- #1292: In the linkchecker, retry HEAD requests when denied by HTTP 405. Also make the redirect code apparent and tweak the output a bit to be more obvious.
- #1285: Avoid name clashes between C domain objects and section titles.

[3] http://docutils.sourceforge.net/docs/ref/rst/directives.html#figure

- #848: Always take the newest code in incremental rebuilds with the *sphinx.ext.viewcode* extension.
- #979, #1266: Fix exclude handling in `sphinx-apidoc`.
- #1302: Fix regression in *sphinx.ext.inheritance_diagram* when documenting classes that can't be pickled.
- #1316: Remove hard-coded `font-face` resources from epub theme.
- #1329: Fix traceback with empty translation msgstr in .po files.
- #1300: Fix references not working in translated documents in some instances.
- #1283: Fix a bug in the detection of changed files that would try to access doctrees of deleted documents.
- #1330: Fix *exclude_patterns* behavior with subdirectories in the *html_static_path*.
- #1323: Fix emitting empty `<ul>` tags in the HTML writer, which is not valid HTML.
- #1147: Don't emit a sidebar search box in the "singlehtml" builder.

20.9.2 Documentation

- #1325: Added a "Intersphinx" tutorial section. (`doc/tutorial.rst`)

20.10 Release 1.2 (released Dec 10, 2013)

20.10.1 Features added

- Added `sphinx.version_info` tuple for programmatic checking of the Sphinx version.

20.10.2 Incompatible changes

- Removed the `sphinx.ext.refcounting` extension – it is very specific to CPython and has no place in the main distribution.

20.10.3 Bugs fixed

- Restore `versionmodified` CSS class for versionadded/changed and deprecated directives.
- PR#181: Fix `html_theme_path = ['.']` is a trigger of rebuild all documents always (This change keeps the current "theme changes cause a rebuild" feature).
- #1296: Fix invalid charset in HTML help generated HTML files for default locale.
- PR#190: Fix gettext does not extract figure caption and rubric title inside other blocks. Thanks to Michael Schlenker.
- PR#176: Make sure setup_command test can always import Sphinx. Thanks to Dmitry Shachnev.
- #1311: Fix test_linkcode.test_html fails with C locale and Python 3.
- #1269: Fix ResourceWarnings with Python 3.2 or later.

- #1138: Fix: When `autodoc_docstring_signature = True` and `autoclass_content = 'init'` or `'both'`, __init__ line should be removed from class documentation.

20.11 Release 1.2 beta3 (released Oct 3, 2013)

20.11.1 Features added

- The Sphinx error log files will now include a list of the loaded extensions for help in debugging.

20.11.2 Incompatible changes

- PR#154: Remove "sphinx" prefix from LaTeX class name except 'sphinxmanual' and 'sphinxhowto'. Now you can use your custom document class without 'sphinx' prefix. Thanks to Erik B.

20.11.3 Bugs fixed

- #1265: Fix i18n: crash when translating a section name that is pointed to from a named target.
- A wrong condition broke the search feature on first page that is usually index.rst. This issue was introduced in 1.2b1.
- #703: When Sphinx can't decode filenames with non-ASCII characters, Sphinx now catches UnicodeError and will continue if possible instead of raising the exception.

20.12 Release 1.2 beta2 (released Sep 17, 2013)

20.12.1 Features added

- `apidoc` now ignores "_private" modules by default, and has an option `-P` to include them.
- `apidoc` now has an option to not generate headings for packages and modules, for the case that the module docstring already includes a reST heading.
- PR#161: `apidoc` can now write each module to a standalone page instead of combining all modules in a package on one page.
- Builders: rebuild i18n target document when catalog updated.
- Support docutils.conf 'writers' and 'html4css1 writer' section in the HTML writer. The latex, manpage and texinfo writers also support their respective 'writers' sections.
- The new *`html_extra_path`* config value allows to specify directories with files that should be copied directly to the HTML output directory.
- Autodoc directives for module data and attributes now support an `annotation` option, so that the default display of the data/attribute value can be overridden.
- PR#136: Autodoc directives now support an `imported-members` option to include members imported from different modules.
- New locales: Macedonian, Sinhala, Indonesian.
- Theme package collection by using setuptools plugin mechanism.

20.12.2 Incompatible changes

- PR#144, #1182: Force timezone offset to LocalTimeZone on POT-Creation-Date that was generated by gettext builder. Thanks to masklinn and Jakub Wilk.

20.12.3 Bugs fixed

- PR#132: Updated jQuery version to 1.8.3.
- PR#141, #982: Avoid crash when writing PNG file using Python 3. Thanks to Marcin Wojdyr.
- PR#145: In parallel builds, sphinx drops second document file to write. Thanks to tychoish.
- PR#151: Some styling updates to tables in LaTeX.
- PR#153: The "extensions" config value can now be overridden.
- PR#155: Added support for some C++11 function qualifiers.
- Fix: 'make gettext' caused UnicodeDecodeError when templates contain utf-8 encoded strings.
- #828: use inspect.getfullargspec() to be able to document functions with keyword-only arguments on Python 3.
- #1090: Fix i18n: multiple cross references (term, ref, doc) in the same line return the same link.
- #1157: Combination of 'globaltoc.html' and hidden toctree caused exception.
- #1159: fix wrong generation of objects inventory for Python modules, and add a workaround in intersphinx to fix handling of affected inventories.
- #1160: Citation target missing caused an AssertionError.
- #1162, PR#139: singlehtml builder didn't copy images to _images/.
- #1173: Adjust setup.py dependencies because Jinja2 2.7 discontinued compatibility with Python < 3.3 and Python < 2.6. Thanks to Alexander Dupuy.
- #1185: Don't crash when a Python module has a wrong or no encoding declared, and non-ASCII characters are included.
- #1188: sphinx-quickstart raises UnicodeEncodeError if "Project version" includes non-ASCII characters.
- #1189: "Title underline is too short" WARNING is given when using fullwidth characters to "Project name" on quickstart.
- #1190: Output TeX/texinfo/man filename has no basename (only extension) when using non-ASCII characters in the "Project name" on quickstart.
- #1192: Fix escaping problem for hyperlinks in the manpage writer.
- #1193: Fix i18n: multiple link references in the same line return the same link.
- #1176: Fix i18n: footnote reference number missing for auto numbered named footnote and auto symbol footnote.
- PR#146,#1172: Fix ZeroDivisionError in parallel builds. Thanks to tychoish.
- #1204: Fix wrong generation of links to local intersphinx targets.
- #1206: Fix i18n: gettext did not translate admonition directive's title.
- #1232: Sphinx generated broken ePub files on Windows.

- #1259: Guard the debug output call when emitting events; to prevent the repr() implementation of arbitrary objects causing build failures.
- #1142: Fix NFC/NFD normalizing problem of rst filename on Mac OS X.
- #1234: Ignoring the string consists only of white-space characters.

20.13 Release 1.2 beta1 (released Mar 31, 2013)

20.13.1 Incompatible changes

- Removed `sphinx.util.compat.directive_dwim()` and `sphinx.roles.xfileref_role()` which were deprecated since version 1.0.
- PR#122: the files given in *`latex_additional_files`* now override TeX files included by Sphinx, such as `sphinx.sty`.
- PR#124: the node generated by *`versionadded`*, *`versionchanged`* and *`deprecated`* directives now includes all added markup (such as "New in version X") as child nodes, and no additional text must be generated by writers.
- PR#99: the *`seealso`* directive now generates admonition nodes instead of the custom `seealso` node.

20.13.2 Features added

- Markup
  - The *`toctree`* directive and the `toctree()` template function now have an `includehidden` option that includes hidden toctree entries (bugs #790 and #1047). A bug in the `maxdepth` option for the `toctree()` template function has been fixed (bug #1046).
  - PR#99: Strip down seealso directives to normal admonitions. This removes their unusual CSS classes (admonition-see-also), inconsistent LaTeX admonition title ("See Also" instead of "See also"), and spurious indentation in the text builder.
- HTML builder
  - #783: Create a link to full size image if it is scaled with width or height.
  - #1067: Improve the ordering of the JavaScript search results: matches in titles come before matches in full text, and object results are better categorized. Also implement a pluggable search scorer.
  - #1053: The "rightsidebar" and "collapsiblesidebar" HTML theme options now work together.
  - Update to jQuery 1.7.1 and Underscore.js 1.3.1.
- Texinfo builder
  - An "Index" node is no longer added when there are no entries.
  - "deffn" categories are no longer capitalized if they contain capital letters.
  - `desc_annotation` nodes are now rendered.
  - `strong` and `emphasis` nodes are now formatted like `literal`s. The reason for this is because the standard Texinfo markup (`*strong*` and `_emphasis_`) resulted in confusing output due to the common usage of using these constructs for documenting parameter names.
  - Field lists formatting has been tweaked to better display "Info field lists".

  - `system_message` and `problematic` nodes are now formatted in a similar fashion as done by the text builder.
  - "en-dash" and "em-dash" conversion of hyphens is no longer performed in option directive signatures.
  - `@ref` is now used instead of `@pxref` for cross-references which prevents the word "see" from being added before the link (does not affect the Info output).
  - The `@finalout` command has been added for better TeX output.
  - `transition` nodes are now formatted using underscores ("_") instead of asterisks ("*").
  - The default value for the `paragraphindent` has been changed from 2 to 0 meaning that paragraphs are no longer indented by default.
  - #1110: A new configuration value *`texinfo_no_detailmenu`* has been added for controlling whether a `@detailmenu` is added in the "Top" node's menu.
  - Detailed menus are no longer created except for the "Top" node.
  - Fixed an issue where duplicate domain indices would result in invalid output.
- LaTeX builder:
  - PR#115: Add `'transition'` item in *`latex_elements`* for customizing how transitions are displayed. Thanks to Jeff Klukas.
  - PR#114: The LaTeX writer now includes the "cmap" package by default. The `'cmappkg'` item in *`latex_elements`* can be used to control this. Thanks to Dmitry Shachnev.
  - The `'fontpkg'` item in *`latex_elements`* now defaults to `''` when the *`language`* uses the Cyrillic script. Suggested by Dmitry Shachnev.
  - The *`latex_documents`*, *`texinfo_documents`*, and *`man_pages`* configuration values will be set to default values based on the *`master_doc`* if not explicitly set in `conf.py`. Previously, if these values were not set, no output would be generated by their respective builders.
- Internationalization:
  - Add i18n capabilities for custom templates. For example: The Sphinx reference documentation in doc directory provides a `sphinx.pot` file with message strings from `doc/_templates/*.html` when using `make gettext`.
  - PR#61,#703: Add support for non-ASCII filename handling.
- Other builders:
  - Added the Docutils-native XML and pseudo-XML builders. See `XMLBuilder` and `PseudoXMLBuilder`.
  - PR#45: The linkcheck builder now checks `#anchor`s for existence.
  - PR#123, #1106: Add *`epub_use_index`* configuration value. If provided, it will be used instead of *`html_use_index`* for epub builder.
  - PR#126: Add *`epub_tocscope`* configuration value. The setting controls the generation of the epub toc. The user can now also include hidden toc entries.
  - PR#112: Add *`epub_show_urls`* configuration value.
- Extensions:
  - PR#52: `special_members` flag to autodoc now behaves like `members`.
  - PR#47: Added *`sphinx.ext.linkcode`* extension.

  - PR#25: In inheritance diagrams, the first line of the class docstring is now the tooltip for the class.

- Command-line interfaces:
  - PR#75: Added `--follow-links` option to sphinx-apidoc.
  - #869: sphinx-build now has the option `-T` for printing the full traceback after an unhandled exception.
  - sphinx-build now supports the standard `--help` and `--version` options.
  - sphinx-build now provides more specific error messages when called with invalid options or arguments.
  - sphinx-build now has a verbose option `-v` which can be repeated for greater effect. A single occurrence provides a slightly more verbose output than normal. Two or more occurrences of this option provides more detailed output which may be useful for debugging.
- Locales:
  - PR#74: Fix some Russian translation.
  - PR#54: Added Norwegian bokmaal translation.
  - PR#35: Added Slovak translation.
  - PR#28: Added Hungarian translation.
  - #1113: Add Hebrew locale.
  - #1097: Add Basque locale.
  - #1037: Fix typos in Polish translation. Thanks to Jakub Wilk.
  - #1012: Update Estonian translation.
- Optimizations:
  - Speed up building the search index by caching the results of the word stemming routines. Saves about 20 seconds when building the Python documentation.
  - PR#108: Add experimental support for parallel building with a new `-j` option.

20.13.3 Documentation

- PR#88: Added the "Sphinx Developer's Guide" (`doc/devguide.rst`) which outlines the basic development process of the Sphinx project.
- Added a detailed "Installing Sphinx" document (`doc/install.rst`).

20.13.4 Bugs fixed

- PR#124: Fix paragraphs in versionmodified are ignored when it has no dangling paragraphs. Fix wrong html output (nested `<p>` tag). Fix versionmodified is not translatable. Thanks to Nozomu Kaneko.
- PR#111: Respect add_autodoc_attrgetter() even when inherited-members is set. Thanks to A. Jesse Jiryu Davis.
- PR#97: Fix footnote handling in translated documents.
- Fix text writer not handling visit_legend for figure directive contents.

- Fix text builder not respecting wide/fullwidth characters: title underline width, table layout width and text wrap width.
- Fix leading space in LaTeX table header cells.
- #1132: Fix LaTeX table output for multi-row cells in the first column.
- #1128: Fix Unicode errors when trying to format time strings with a non-standard locale.
- #1127: Fix traceback when autodoc tries to tokenize a non-Python file.
- #1126: Fix double-hyphen to en-dash conversion in wrong places such as command-line option names in LaTeX.
- #1123: Allow whitespaces in filenames given to *literalinclude*.
- #1120: Added improvements about i18n for themes "basic", "haiku" and "scrolls" that Sphinx built-in. Thanks to Leonardo J. Caballero G.
- #1118: Updated Spanish translation. Thanks to Leonardo J. Caballero G.
- #1117: Handle .pyx files in sphinx-apidoc.
- #1112: Avoid duplicate download files when referenced from documents in different ways (absolute/relative).
- #1111: Fix failure to find uppercase words in search when *html_search_language* is 'ja'. Thanks to Tomo Saito.
- #1108: The text writer now correctly numbers enumerated lists with non-default start values (based on patch by Ewan Edwards).
- #1102: Support multi-context "with" statements in autodoc.
- #1090: Fix gettext not extracting glossary terms.
- #1074: Add environment version info to the generated search index to avoid compatibility issues with old builds.
- #1070: Avoid un-pickling issues when running Python 3 and the saved environment was created under Python 2.
- #1069: Fixed error caused when autodoc would try to format signatures of "partial" functions without keyword arguments (patch by Artur Gaspar).
- #1062: sphinx.ext.autodoc use __init__ method signature for class signature.
- #1055: Fix web support with relative path to source directory.
- #1043: Fix sphinx-quickstart asking again for yes/no questions because `input()` returns values with an extra 'r' on Python 3.2.0 + Windows. Thanks to Régis Décamps.
- #1041: Fix failure of the cpp domain parser to parse a const type with a modifier.
- #1038: Fix failure of the cpp domain parser to parse C+11 "static constexpr" declarations. Thanks to Jakub Wilk.
- #1029: Fix intersphinx_mapping values not being stable if the mapping has plural key/value set with Python 3.3.
- #1028: Fix line block output in the text builder.
- #1024: Improve Makefile/make.bat error message if Sphinx is not found. Thanks to Anatoly Techtonik.
- #1018: Fix "container" directive handling in the text builder.

- #1015: Stop overriding jQuery contains() in the JavaScript.
- #1010: Make pngmath images transparent by default; IE7+ should handle it.
- #1008: Fix test failures with Python 3.3.
- #995: Fix table-of-contents and page numbering for the LaTeX "howto" class.
- #976: Fix gettext does not extract index entries.
- PR#72: #975: Fix gettext not extracting definition terms before docutils 0.10.
- #961: Fix LaTeX output for triple quotes in code snippets.
- #958: Do not preserve `environment.pickle` after a failed build.
- #955: Fix i18n transformation.
- #940: Fix gettext does not extract figure caption.
- #920: Fix PIL packaging issue that allowed to import `Image` without PIL namespace. Thanks to Marc Schlaich.
- #723: Fix the search function on local files in WebKit based browsers.
- #440: Fix coarse timestamp resolution in some filesystem generating a wrong list of outdated files.

20.14 Release 1.1.3 (Mar 10, 2012)

- PR#40: Fix `safe_repr` function to decode bytestrings with non-ASCII characters correctly.
- PR#37: Allow configuring sphinx-apidoc via `SPHINX_APIDOC_OPTIONS`.
- PR#34: Restore Python 2.4 compatibility.
- PR#36: Make the "bibliography to TOC" fix in LaTeX output specific to the document class.
- #695: When the highlight language "python" is specified explicitly, do not try to parse the code to recognize non-Python snippets.
- #859: Fix exception under certain circumstances when not finding appropriate objects to link to.
- #860: Do not crash when encountering invalid doctest examples, just emit a warning.
- #864: Fix crash with some settings of *`modindex_common_prefix`*.
- #862: Fix handling of `-D` and `-A` options on Python 3.
- #851: Recognize and warn about circular toctrees, instead of running into recursion errors.
- #853: Restore compatibility with docutils trunk.
- #852: Fix HtmlHelp index entry links again.
- #854: Fix inheritance_diagram raising attribute errors on builtins.
- #832: Fix crashes when putting comments or lone terms in a glossary.
- #834, #818: Fix HTML help language/encoding mapping for all Sphinx supported languages.
- #844: Fix crashes when dealing with Unicode output in doctest extension.
- #831: Provide `--project` flag in setup_command as advertised.
- #875: Fix reading config files under Python 3.
- #876: Fix quickstart test under Python 3.

- #870: Fix spurious KeyErrors when removing documents.
- #892: Fix single-HTML builder misbehaving with the master document in a subdirectory.
- #873: Fix assertion errors with empty `only` directives.
- #816: Fix encoding issues in the Qt help builder.

20.15 Release 1.1.2 (Nov 1, 2011) – 1.1.1 is a silly version number anyway!

- #809: Include custom fixers in the source distribution.

20.16 Release 1.1.1 (Nov 1, 2011)

- #791: Fix QtHelp, DevHelp and HtmlHelp index entry links.
- #792: Include "sphinx-apidoc" in the source distribution.
- #797: Don't crash on a misformatted glossary.
- #801: Make intersphinx work properly without SSL support.
- #805: Make the `Sphinx.add_index_to_domain` method work correctly.
- #780: Fix Python 2.5 compatibility.

20.17 Release 1.1 (Oct 9, 2011)

20.17.1 Incompatible changes

- The *`py:module`* directive doesn't output its `platform` option value anymore. (It was the only thing that the directive did output, and therefore quite inconsistent.)
- Removed support for old dependency versions; requirements are now:
  - Pygments >= 1.2
  - Docutils >= 0.7
  - Jinja2 >= 2.3

20.17.2 Features added

- Added Python 3.x support.
- New builders and subsystems:
  - Added a Texinfo builder.
  - Added i18n support for content, a `gettext` builder and related utilities.
  - Added the `websupport` library and builder.
  - #98: Added a `sphinx-apidoc` script that autogenerates a hierarchy of source files containing autodoc directives to document modules and packages.

  - #273: Add an API for adding full-text search support for languages other than English. Add support for Japanese.

- Markup:
  - #138: Added an *index* role, to make inline index entries.
  - #454: Added more index markup capabilities: marking see/seealso entries, and main entries for a given key.
  - #460: Allowed limiting the depth of section numbers for HTML using the *toctree*'s `numbered` option.
  - #586: Implemented improved *glossary* markup which allows multiple terms per definition.
  - #478: Added *py:decorator* directive to describe decorators.
  - C++ domain now supports array definitions.
  - C++ domain now supports doc fields (`:param x:` inside directives).
  - Section headings in *only* directives are now correctly handled.
  - Added `emphasize-lines` option to source code directives.
  - #678: C++ domain now supports superclasses.
- HTML builder:
  - Added `pyramid` theme.
  - #559: *html_add_permalinks* is now a string giving the text to display in permalinks.
  - #259: HTML table rows now have even/odd CSS classes to enable "Zebra styling".
  - #554: Add theme option `sidebarwidth` to the basic theme.
- Other builders:
  - #516: Added new value of the *latex_show_urls* option to show the URLs in footnotes.
  - #209: Added *text_newlines* and *text_sectionchars* config values.
  - Added *man_show_urls* config value.
  - #472: linkcheck builder: Check links in parallel, use HTTP HEAD requests and allow configuring the timeout. New config values: *linkcheck_timeout* and *linkcheck_workers*.
  - #521: Added *linkcheck_ignore* config value.
  - #28: Support row/colspans in tables in the LaTeX builder.
- Configuration and extensibility:
  - #537: Added *nitpick_ignore*.
  - #306: Added *env-get-outdated* event.
  - `Application.add_stylesheet()` now accepts full URIs.
- Autodoc:
  - #564: Add *autodoc_docstring_signature*. When enabled (the default), autodoc retrieves the signature from the first line of the docstring, if it is found there.
  - #176: Provide `private-members` option for autodoc directives.
  - #520: Provide `special-members` option for autodoc directives.
  - #431: Doc comments for attributes can now be given on the same line as the assignment.

  - #437: autodoc now shows values of class data attributes.
  - autodoc now supports documenting the signatures of `functools.partial` objects.
- Other extensions:
  - Added the *sphinx.ext.mathjax* extension.
  - #443: Allow referencing external graphviz files.
  - Added `inline` option to graphviz directives, and fixed the default (block-style) in LaTeX output.
  - #590: Added `caption` option to graphviz directives.
  - #553: Added *testcleanup* blocks in the doctest extension.
  - #594: *trim_doctest_flags* now also removes `<BLANKLINE>` indicators.
  - #367: Added automatic exclusion of hidden members in inheritance diagrams, and an option to selectively enable it.
  - Added *pngmath_add_tooltips*.
  - The math extension displaymath directives now support `name` in addition to `label` for giving the equation label, for compatibility with Docutils.
- New locales:
  - #221: Added Swedish locale.
  - #526: Added Iranian locale.
  - #694: Added Latvian locale.
  - Added Nepali locale.
  - #714: Added Korean locale.
  - #766: Added Estonian locale.
- Bugs fixed:
  - #778: Fix "hide search matches" link on pages linked by search.
  - Fix the source positions referenced by the "viewcode" extension.

20.18 Release 1.0.8 (Sep 23, 2011)

- #627: Fix tracebacks for AttributeErrors in autosummary generation.
- Fix the `abbr` role when the abbreviation has newlines in it.
- #727: Fix the links to search results with custom object types.
- #648: Fix line numbers reported in warnings about undefined references.
- #696, #666: Fix C++ array definitions and template arguments that are not type names.
- #633: Allow footnotes in section headers in LaTeX output.
- #616: Allow keywords to be linked via intersphinx.
- #613: Allow Unicode characters in production list token names.
- #720: Add dummy visitors for graphviz nodes for text and man.
- #704: Fix image file duplication bug.

- #677: Fix parsing of multiple signatures in C++ domain.
- #637: Ignore Emacs lock files when looking for source files.
- #544: Allow .pyw extension for importable modules in autodoc.
- #700: Use `$(MAKE)` in quickstart-generated Makefiles.
- #734: Make sidebar search box width consistent in browsers.
- #644: Fix spacing of centered figures in HTML output.
- #767: Safely encode SphinxError messages when printing them to sys.stderr.
- #611: Fix LaTeX output error with a document with no sections but a link target.
- Correctly treat built-in method descriptors as methods in autodoc.
- #706: Stop monkeypatching the Python textwrap module.
- #657: viewcode now works correctly with source files that have non-ASCII encoding.
- #669: Respect the `noindex` flag option in py:module directives.
- #675: Fix IndexErrors when including nonexisting lines with *`literalinclude`*.
- #676: Respect custom function/method parameter separator strings.
- #682: Fix JS incompatibility with jQuery >= 1.5.
- #693: Fix double encoding done when writing HTMLHelp .hhk files.
- #647: Do not apply SmartyPants in parsed-literal blocks.
- C++ domain now supports array definitions.

20.19 Release 1.0.7 (Jan 15, 2011)

- #347: Fix wrong generation of directives of static methods in autosummary.
- #599: Import PIL as `from PIL import Image`.
- #558: Fix longtables with captions in LaTeX output.
- Make token references work as hyperlinks again in LaTeX output.
- #572: Show warnings by default when reference labels cannot be found.
- #536: Include line number when complaining about missing reference targets in nitpicky mode.
- #590: Fix inline display of graphviz diagrams in LaTeX output.
- #589: Build using app.build() in setup command.
- Fix a bug in the inheritance diagram exception that caused base classes to be skipped if one of them is a builtin.
- Fix general index links for C++ domain objects.
- #332: Make admonition boundaries in LaTeX output visible.
- #573: Fix KeyErrors occurring on rebuild after removing a file.
- Fix a traceback when removing files with globbed toctrees.
- If an autodoc object cannot be imported, always re-read the document containing the directive on next build.

- If an autodoc object cannot be imported, show the full traceback of the import error.
- Fix a bug where the removal of download files and images wasn't noticed.
- #571: Implement ~ cross-reference prefix for the C domain.
- Fix regression of LaTeX output with the fix of #556.
- #568: Fix lookup of class attribute documentation on descriptors so that comment documentation now works.
- Fix traceback with `only` directives preceded by targets.
- Fix tracebacks occurring for duplicate C++ domain objects.
- Fix JavaScript domain links to objects with `$` in their name.

20.20 Release 1.0.6 (Jan 04, 2011)

- #581: Fix traceback in Python domain for empty cross-reference targets.
- #283: Fix literal block display issues on Chrome browsers.
- #383, #148: Support sorting a limited range of accented characters in the general index and the glossary.
- #570: Try decoding `-D` and `-A` command-line arguments with the locale's preferred encoding.
- #528: Observe *`locale_dirs`* when looking for the JS translations file.
- #574: Add special code for better support of Japanese documents in the LaTeX builder.
- Regression of #77: If there is only one parameter given with `:param:` markup, the bullet list is now suppressed again.
- #556: Fix missing paragraph breaks in LaTeX output in certain situations.
- #567: Emit the `autodoc-process-docstring` event even for objects without a docstring so that it can add content.
- #565: In the LaTeX builder, not only literal blocks require different table handling, but also quite a few other list-like block elements.
- #515: Fix tracebacks in the viewcode extension for Python objects that do not have a valid signature.
- Fix strange reports of line numbers for warnings generated from autodoc-included docstrings, due to different behavior depending on docutils version.
- Several fixes to the C++ domain.

20.21 Release 1.0.5 (Nov 12, 2010)

- #557: Add CSS styles required by docutils 0.7 for aligned images and figures.
- In the Makefile generated by LaTeX output, do not delete pdf files on clean; they might be required images.
- #535: Fix LaTeX output generated for line blocks.
- #544: Allow `.pyw` as a source file extension.

20.22 Release 1.0.4 (Sep 17, 2010)

- #524: Open intersphinx inventories in binary mode on Windows, since version 2 contains zlib-compressed data.
- #513: Allow giving non-local URIs for JavaScript files, e.g. in the JSMath extension.
- #512: Fix traceback when `intersphinx_mapping` is empty.

20.23 Release 1.0.3 (Aug 23, 2010)

- #495: Fix internal vs. external link distinction for links coming from a docutils table-of-contents.
- #494: Fix the `maxdepth` option for the `toctree()` template callable when used with `collapse=True`.
- #507: Fix crash parsing Python argument lists containing brackets in string literals.
- #501: Fix regression when building LaTeX docs with figures that don't have captions.
- #510: Fix inheritance diagrams for classes that are not picklable.
- #497: Introduce separate background color for the sidebar collapse button, making it easier to see.
- #502, #503, #496: Fix small layout bugs in several builtin themes.

20.24 Release 1.0.2 (Aug 14, 2010)

- #490: Fix cross-references to objects of types added by the `add_object_type()` API function.
- Fix handling of doc field types for different directive types.
- Allow breaking long signatures, continuing with backlash-escaped newlines.
- Fix unwanted styling of C domain references (because of a namespace clash with Pygments styles).
- Allow references to PEPs and RFCs with explicit anchors.
- #471: Fix LaTeX references to figures.
- #482: When doing a non-exact search, match only the given type of object.
- #481: Apply non-exact search for Python reference targets with `.name` for modules too.
- #484: Fix crash when duplicating a parameter in an info field list.
- #487: Fix setting the default role to one provided by the `oldcmarkup` extension.
- #488: Fix crash when json-py is installed, which provides a `json` module but is incompatible to simplejson.
- #480: Fix handling of target naming in intersphinx.
- #486: Fix removal of `!` for all cross-reference roles.

20.25 Release 1.0.1 (Jul 27, 2010)

- #470: Fix generated target names for reST domain objects; they are not in the same namespace.
- #266: Add Bengali language.
- #473: Fix a bug in parsing JavaScript object names.
- #474: Fix building with SingleHTMLBuilder when there is no toctree.
- Fix display names for objects linked to by intersphinx with explicit targets.
- Fix building with the JSON builder.
- Fix hyperrefs in object descriptions for LaTeX.

20.26 Release 1.0 (Jul 23, 2010)

20.26.1 Incompatible changes

- Support for domains has been added. A domain is a collection of directives and roles that all describe objects belonging together, e.g. elements of a programming language. A few builtin domains are provided:
  - Python
  - C
  - C++
  - JavaScript
  - reStructuredText
- The old markup for defining and linking to C directives is now deprecated. It will not work anymore in future versions without activating the `oldcmarkup` extension; in Sphinx 1.0, it is activated by default.
- Removed support for old dependency versions; requirements are now:
  - docutils >= 0.5
  - Jinja2 >= 2.2
- Removed deprecated elements:
  - `exclude_dirs` config value
  - `sphinx.builder` module

20.26.2 Features added

- General:
  - Added a "nitpicky" mode that emits warnings for all missing references. It is activated by the `-n` command-line switch or the *`nitpicky`* config value.
  - Added `latexpdf` target in quickstart Makefile.
- Markup:

  - The *menuselection* and *guilabel* roles now support ampersand accelerators.
  - New more compact doc field syntax is now recognized: `:param type name: description`.
  - Added `tab-width` option to *literalinclude* directive.
  - Added `titlesonly` option to *toctree* directive.
  - Added the `prepend` and `append` options to the *literalinclude* directive.
  - #284: All docinfo metadata is now put into the document metadata, not just the author.
  - The *ref* role can now also reference tables by caption.
  - The include[4] directive now supports absolute paths, which are interpreted as relative to the source directory.
  - In the Python domain, references like `:func:`.name`` now look for matching names with any prefix if no direct match is found.
- Configuration:
  - Added *rst_prolog* config value.
  - Added *html_secnumber_suffix* config value to control section numbering format.
  - Added *html_compact_lists* config value to control docutils' compact lists feature.
  - The *html_sidebars* config value can now contain patterns as keys, and the values can be lists that explicitly select which sidebar templates should be rendered. That means that the builtin sidebar contents can be included only selectively.
  - *html_static_path* can now contain single file entries.
  - The new universal config value *exclude_patterns* makes the old `unused_docs`, `exclude_trees` and `exclude_dirnames` obsolete.
  - Added *html_output_encoding* config value.
  - Added the *latex_docclass* config value and made the "twoside" documentclass option overridable by "oneside".
  - Added the *trim_doctest_flags* config value, which is true by default.
  - Added *html_show_copyright* config value.
  - Added *latex_show_pagerefs* and *latex_show_urls* config values.
  - The behavior of *html_file_suffix* changed slightly: the empty string now means "no suffix" instead of "default suffix", use `None` for "default suffix".
- New builders:
  - Added a builder for the Epub format.
  - Added a builder for manual pages.
  - Added a single-file HTML builder.
- HTML output:
  - Inline roles now get a CSS class with their name, allowing styles to customize their appearance. Domain-specific roles get two classes, `domain` and `domain-rolename`.

[4] http://docutils.sourceforge.net/docs/ref/rst/directives.html#include

  - References now get the class `internal` if they are internal to the whole project, as opposed to internal to the current page.
  - External references can be styled differently with the new `externalrefs` theme option for the default theme.
  - In the default theme, the sidebar can experimentally now be made collapsible using the new `collapsiblesidebar` theme option.
  - #129: Toctrees are now wrapped in a `div` tag with class `toctree-wrapper` in HTML output.
  - The *`toctree`* callable in templates now has a `maxdepth` keyword argument to control the depth of the generated tree.
  - The *`toctree`* callable in templates now accepts a `titles_only` keyword argument.
  - Added `htmltitle` block in layout template.
  - In the JavaScript search, allow searching for object names including the module name, like `sys.argv`.
  - Added new theme `haiku`, inspired by the Haiku OS user guide.
  - Added new theme `nature`.
  - Added new theme `agogo`, created by Andi Albrecht.
  - Added new theme `scrolls`, created by Armin Ronacher.
  - #193: Added a `visitedlinkcolor` theme option to the default theme.
  - #322: Improved responsiveness of the search page by loading the search index asynchronously.
- Extension API:
  - Added *`html-collect-pages`*.
  - Added *`needs_sphinx`* config value and *`require_sphinx()`* application API method.
  - #200: Added *`add_stylesheet()`* application API method.
- Extensions:
  - Added the *`viewcode`* extension.
  - Added the *`extlinks`* extension.
  - Added support for source ordering of members in autodoc, with `autodoc_member_order = 'bysource'`.
  - Added *`autodoc_default_flags`* config value, which can be used to select default flags for all autodoc directives.
  - Added a way for intersphinx to refer to named labels in other projects, and to specify the project you want to link to.
  - #280: Autodoc can now document instance attributes assigned in `__init__` methods.
  - Many improvements and fixes to the *`autosummary`* extension, thanks to Pauli Virtanen.
  - #309: The *`graphviz`* extension can now output SVG instead of PNG images, controlled by the *`graphviz_output_format`* config value.
  - Added `alt` option to *`graphviz`* extension directives.
  - Added `exclude` argument to *`autodoc.between()`*.
- Translations:

  - Added Croatian translation, thanks to Bojan Mihelač.
  - Added Turkish translation, thanks to Firat Ozgul.
  - Added Catalan translation, thanks to Pau Fernández.
  - Added simplified Chinese translation.
  - Added Danish translation, thanks to Hjorth Larsen.
  - Added Lithuanian translation, thanks to Dalius Dobravolskas.
- Bugs fixed:
  - #445: Fix links to result pages when using the search function of HTML built with the `dirhtml` builder.
  - #444: In templates, properly re-escape values treated with the "striptags" Jinja filter.

20.27 Previous versions

The changelog for versions before 1.0 can be found in the file `CHANGES.old` in the source distribution or at Github[5].

[5] https://github.com/sphinx-doc/sphinx/raw/master/CHANGES.old

CHAPTER 21

Projects using Sphinx

This is an (incomplete) alphabetic list of projects that use Sphinx or are experimenting with using it for their documentation. If you like to be included, please mail to the Google group[1].

I've grouped the list into sections to make it easier to find interesting examples.

21.1 Documentation using the default theme

- APSW: http://apidoc.apsw.googlecode.com/hg/index.html
- ASE: https://wiki.fysik.dtu.dk/ase/
- Calibre: http://manual.calibre-ebook.com/
- CodePy: http://documen.tician.de/codepy/
- Cython: http://docs.cython.org/
- Cormoran: http://cormoran.nhopkg.org/docs/
- Director: http://pythonhosted.org/director/
- Dirigible: http://www.projectdirigible.com/documentation/
- F2py: http://f2py.sourceforge.net/docs/
- GeoDjango: https://docs.djangoproject.com/en/dev/ref/contrib/gis/
- Genomedata: http://noble.gs.washington.edu/proj/genomedata/doc/1.2.2/genomedata.html
- gevent: http://www.gevent.org/
- Google Wave API: http://wave-robot-python-client.googlecode.com/svn/trunk/pydocs/index.html
- GSL Shell: http://www.nongnu.org/gsl-shell/
- Heapkeeper: http://heapkeeper.org/
- Hands-on Python Tutorial: http://anh.cs.luc.edu/python/hands-on/3.1/handsonHtml/
- Hedge: http://documen.tician.de/hedge/
- Leo: http://leoeditor.com/
- Lino: http://lino.saffre-rumma.net/
- MeshPy: http://documen.tician.de/meshpy/

[1] https://groups.google.com/group/sphinx-users

- mpmath: http://mpmath.googlecode.com/svn/trunk/doc/build/index.html
- OpenEXR: http://excamera.com/articles/26/doc/index.html
- OpenGDA: http://www.opengda.org/gdadoc/html/
- openWNS: http://docs.openwns.org/
- Paste: http://pythonpaste.org/script/
- Paver: http://paver.github.io/paver/
- Pioneers and Prominent Men of Utah: http://pioneers.rstebbing.com/
- PyCantonese: http://pycantonese.github.io/
- Pyccuracy: https://github.com/heynemann/pyccuracy/wiki/
- PyCuda: http://documen.tician.de/pycuda/
- Pyevolve: http://pyevolve.sourceforge.net/
- Pylo: http://documen.tician.de/pylo/
- PyMQI: http://pythonhosted.org/pymqi/
- PyPubSub: http://pubsub.sourceforge.net/
- pySPACE: http://pyspace.github.io/pyspace/
- Python: https://docs.python.org/
- python-apt: http://apt.alioth.debian.org/python-apt-doc/
- PyUblas: http://documen.tician.de/pyublas/
- Quex: http://quex.sourceforge.net/doc/html/main.html
- Scapy: http://www.secdev.org/projects/scapy/doc/
- Segway: http://noble.gs.washington.edu/proj/segway/doc/1.1.0/segway.html
- SimPy: http://simpy.readthedocs.org/en/latest/
- SymPy: http://docs.sympy.org/
- WTForms: http://wtforms.simplecodes.com/docs/
- z3c: http://www.ibiblio.org/paulcarduner/z3ctutorial/

21.2 Documentation using a customized version of the default theme

- Advanced Generic Widgets: http://xoomer.virgilio.it/infinity77/AGW_Docs/index.html
- Bazaar: http://doc.bazaar.canonical.com/en/
- CakePHP: http://book.cakephp.org/2.0/en/index.html
- Chaco: http://docs.enthought.com/chaco/
- Chef: https://docs.chef.io/index.html
- Djagios: http://djagios.org/
- EZ-Draw: http://pageperso.lif.univ-mrs.fr/~edouard.thiel/ez-draw/doc/en/html/ez-manual.html
- GetFEM++: http://home.gna.org/getfem/

- Google or-tools: https://or-tools.googlecode.com/svn/trunk/documentation/user_manual/index.html
- GPAW: https://wiki.fysik.dtu.dk/gpaw/
- Grok: http://grok.zope.org/doc/current/
- IFM: http://fluffybunny.memebot.com/ifm-docs/index.html
- Kaa: http://api.freevo.org/kaa-base/
- LEPL: http://www.acooke.org/lepl/
- Mayavi: http://docs.enthought.com/mayavi/mayavi/
- NICOS: http://trac.frm2.tum.de/nicos/doc/nicos-master/index.html
- NOC: http://redmine.nocproject.org/projects/noc
- NumPy: http://docs.scipy.org/doc/numpy/reference/
- OpenCV: http://docs.opencv.org/
- Peach^3: http://peach3.nl/doc/latest/userdoc/
- Sage: http://sagemath.org/doc/
- SciPy: http://docs.scipy.org/doc/scipy/reference/
- simuPOP: http://simupop.sourceforge.net/manual_release/build/userGuide.html
- Sprox: http://sprox.org/
- TurboGears: http://turbogears.org/2.0/docs/
- Varnish: https://www.varnish-cache.org/docs/
- Zentyal: http://doc.zentyal.org/
- Zope: http://docs.zope.org/zope2/index.html
- zc.async: http://pythonhosted.org/zc.async/1.5.0/

21.3 Documentation using the sphinxdoc theme

- Fityk: http://fityk.nieto.pl/
- MapServer: http://mapserver.org/
- Matplotlib: http://matplotlib.org/
- Music21: http://web.mit.edu/music21/doc/index.html
- NetworkX: http://networkx.github.io/
- Pweave: http://mpastell.com/pweave/
- Pyre: http://docs.danse.us/pyre/sphinx/
- Pysparse: http://pysparse.sourceforge.net/
- PyTango: http://www.esrf.eu/computing/cs/tango/tango_doc/kernel_doc/pytango/latest/index.html
- Python Wild Magic: http://vmlaker.github.io/pythonwildmagic/
- Reteisi: http://www.reteisi.org/contents.html
- Sqlkit: http://sqlkit.argolinux.org/

- Tau: http://www.tango-controls.org/static/tau/latest/doc/html/index.html
- Turbulenz: http://docs.turbulenz.com/
- WebFaction: http://docs.webfaction.com/

21.4 Documentation using another builtin theme

- C/C++ Development with Eclipse: http://eclipsebook.in/ (agogo)
- Jinja: http://jinja.pocoo.org/ (scrolls)
- jsFiddle: http://doc.jsfiddle.net/ (nature)
- libLAS: http://www.liblas.org/ (nature)
- MPipe: http://vmlaker.github.io/mpipe/ (sphinx13)
- pip: https://pip.pypa.io/en/latest/ (sphinx_rtd_theme)
- Pyramid web framework: http://docs.pylonsproject.org/projects/pyramid/en/latest/ (pyramid)
- Programmieren mit PyGTK und Glade (German): http://www.florian-diesch.de/doc/python-und-glade/online/ (agogo)
- Satchmo: http://docs.satchmoproject.com/en/latest/ (sphinx_rtd_theme)
- Setuptools: http://pythonhosted.org/setuptools/ (nature)
- Spring Python: http://docs.spring.io/spring-python/1.2.x/sphinx/html/ (nature)
- sqlparse: http://python-sqlparse.googlecode.com/svn/docs/api/index.html (agogo)
- Sylli: http://sylli.sourceforge.net/ (nature)
- Tuleap Open ALM: https://tuleap.net/doc/en/ (nature)
- Valence: http://docs.valence.desire2learn.com/ (haiku)

21.5 Documentation using a custom theme/integrated in a site

- Blender: http://www.blender.org/documentation/250PythonDoc/
- Blinker: http://discorporate.us/projects/Blinker/docs/
- Ceph: http://ceph.com/docs/master/
- Classy: http://www.pocoo.org/projects/classy/
- DEAP: http://deap.gel.ulaval.ca/doc/0.8/index.html
- Django: http://docs.djangoproject.com/
- Elemental: http://libelemental.org/documentation/dev/index.html
- Enterprise Toolkit for Acrobat products: http://www.adobe.com/devnet-docs/acrobatetk/
- e-cidadania: http://e-cidadania.readthedocs.org/en/latest/
- Flask: http://flask.pocoo.org/docs/
- Flask-OpenID: http://pythonhosted.org/Flask-OpenID/
- Gameduino: http://excamera.com/sphinx/gameduino/

- GeoServer: http://docs.geoserver.org/
- Glashammer: http://glashammer.org/
- Istihza (Turkish Python documentation project): http://www.istihza.com/py2/icindekiler_python.html
- Lasso: http://lassoguide.com/
- Manage documentation such as source code (fr): http://redaction-technique.org/
- MathJax: http://docs.mathjax.org/en/latest/
- MirrorBrain: http://mirrorbrain.org/docs/
- MyHDL: http://docs.myhdl.org/en/latest/
- nose: http://nose.readthedocs.org/en/latest/
- NoTex: https://notex.ch/overview/
- ObjectListView: http://objectlistview.sourceforge.net/python/
- Open ERP: http://doc.openerp.com/
- OpenCV: http://docs.opencv.org/
- Open Dylan: http://opendylan.org/documentation/ and also provides a dylan domain[2]
- OpenLayers: http://docs.openlayers.org/
- PyEphem: http://rhodesmill.org/pyephem/
- German Plone user manual: http://www.hasecke.com/plone-benutzerhandbuch/
- PSI4: http://sirius.chem.vt.edu/psi4manual/latest/index.html
- Pylons: http://docs.pylonsproject.org/projects/pylons-webframework/en/latest/
- PyMOTW: http://pymotw.com/2/
- python-aspectlib: http://python-aspectlib.readthedocs.org/en/latest/ (sphinx-py3doc-enhanced-theme[3])
- QGIS: http://qgis.org/en/docs/index.html
- qooxdoo: http://manual.qooxdoo.org/current/
- Roundup: http://www.roundup-tracker.org/
- Selenium: http://docs.seleniumhq.org/docs/
- Self: http://selflanguage.org/
- Substance D: http://docs.pylonsproject.org/projects/substanced/en/latest/
- Tablib: http://tablib.org/
- SQLAlchemy: http://www.sqlalchemy.org/docs/
- tinyTiM: http://tinytim.sourceforge.net/docs/2.0/
- Ubuntu packaging guide: http://packaging.ubuntu.com/html/
- Werkzeug: http://werkzeug.pocoo.org/docs/
- WFront: http://discorporate.us/projects/WFront/

[2] https://github.com/dylan-lang/sphinx-extensions/blob/master/dylandomain/reference.rst
[3] https://pypi.python.org/pypi/sphinx_py3doc_enhanced_theme

21.6 Homepages and other non-documentation sites

- Applied Mathematics at the Stellenbosch University: http://dip.sun.ac.za/
- A personal page: http://www.dehlia.in/
- Benoit Boissinot: http://bboissin.appspot.com/
- lunarsite: http://lunaryorn.de/
- Red Hot Chili Python: http://redhotchilipython.com/
- The Wine Cellar Book: http://www.thewinecellarbook.com/doc/en/
- UC Berkeley Advanced Control Systems course: http://www.me.berkeley.edu/ME233/sp14/
- VOR: http://www.vor-cycling.be/

21.7 Books produced using Sphinx

- "The `repoze.bfg` Web Application Framework": http://www.amazon.com/repoze-bfg-Web-Application-Framework-Version/dp/0615345379
- A Theoretical Physics Reference book: http://theoretical-physics.net/
- "Simple and Steady Way of Learning for Software Engineering" (in Japanese): http://www.amazon.co.jp/dp/477414259X/
- "Expert Python Programming": http://www.packtpub.com/expert-python-programming/book
- "Expert Python Programming" (Japanese translation): http://www.amazon.co.jp/dp/4048686291/
- "Pomodoro Technique Illustrated" (Japanese translation): http://www.amazon.co.jp/dp/4048689525/
- "Python Professional Programming" (in Japanese): http://www.amazon.co.jp/dp/4798032948/
- "Die Wahrheit des Sehens. Der DEKALOG von Krzysztof Kieślowski": http://www.hasecke.eu/Dekalog/
- The "Varnish Book": https://www.varnish-software.com/static/book/
- "Learning Sphinx" (in Japanese): http://www.oreilly.co.jp/books/9784873116488/
- "LassoGuide": http://www.lassosoft.com/Lasso-Documentation
- "Software-Dokumentation mit Sphinx": http://www.amazon.de/dp/1497448689/

21.8 Thesis using Sphinx

- "A Web-Based System for Comparative Analysis of OpenStreetMap Data by the Use of CouchDB": https://www.yumpu.com/et/document/view/11722645/masterthesis-markusmayr-0542042

CHAPTER 22

Sphinx authors

Sphinx is written and maintained by Georg Brandl <georg@python.org[1]>.

Substantial parts of the templates were written by Armin Ronacher <armin.ronacher@active-4.com[2]>.

Other co-maintainers:

- Takayuki Shimizukawa <shimizukawa@gmail.com[3]>
- Daniel Neuhäuser <@DasIch>
- Jon Waltman <@jonwaltman>
- Rob Ruana <@RobRuana>
- Robert Lehmann <@lehmannro>
- Roland Meister <@rolmei>

Other contributors, listed alphabetically, are:

- Alastair Houghton – Apple Help builder
- Andi Albrecht – agogo theme
- Jakob Lykke Andersen – Rewritten C++ domain
- Henrique Bastos – SVG support for graphviz extension
- Daniel Bültmann – todo extension
- Etienne Desautels – apidoc module
- Michael Droettboom – inheritance_diagram extension
- Charles Duffy – original graphviz extension
- Kevin Dunn – MathJax extension
- Josip Dzolonga – coverage builder
- Hernan Grecco – search improvements
- Horst Gutmann – internationalization support
- Martin Hans – autodoc improvements
- Doug Hellmann – graphviz improvements

[1] georg@python.org
[2] armin.ronacher@active-4.com
[3] shimizukawa@gmail.com

- Takeshi Komiya – numref feature
- Dave Kuhlman – original LaTeX writer
- Blaise Laflamme – pyramid theme
- Thomas Lamb – linkcheck builder
- Łukasz Langa – partial support for autodoc
- Ian Lee – quickstart improvements
- Robert Lehmann – gettext builder (GSOC project)
- Dan MacKinlay – metadata fixes
- Martin Mahner – nature theme
- Will Maier – directory HTML builder
- Jacob Mason – websupport library (GSOC project)
- Roland Meister – epub builder
- Ezio Melotti – collapsible sidebar JavaScript
- Daniel Neuhäuser – JavaScript domain, Python 3 support (GSOC)
- Christopher Perkins – autosummary integration
- Benjamin Peterson – unittests
- 20. Powers – HTML output improvements
- Jeppe Pihl – literalinclude improvements
- Rob Ruana – napoleon extension
- Stefan Seefeld – toctree improvements
- Shibukawa Yoshiki – pluggable search API and Japanese search
- Antonio Valentino – qthelp builder
- Pauli Virtanen – autodoc improvements, autosummary extension
- Stefan van der Walt – autosummary extension
- Thomas Waldmann – apidoc module fixes
- John Waltman – Texinfo builder
- Barry Warsaw – setup command improvements
- Sebastian Wiesner – image handling, distutils support
- Joel Wurtz – cellspanning support in LaTeX

Many thanks for all contributions!

There are also a few modules or functions incorporated from other authors and projects:

- sphinx.util.jsdump uses the basestring encoding from simplejson, written by Bob Ippolito, released under the MIT license
- sphinx.util.stemmer was written by Vivake Gupta, placed in the Public Domain

a

sphinx.addnodes, 165
sphinx.application, 149

b

sphinx.builders, 63
sphinx.builders.applehelp, 64
sphinx.builders.changes, 66
sphinx.builders.devhelp, 64
sphinx.builders.epub, 64
sphinx.builders.gettext, 66
sphinx.builders.html, 63
sphinx.builders.htmlhelp, 63
sphinx.builders.latex, 64
sphinx.builders.linkcheck, 67
sphinx.builders.manpage, 65
sphinx.builders.qthelp, 63
sphinx.builders.texinfo, 65
sphinx.builders.text, 65
sphinx.builders.xml, 67

c

conf, 69
sphinx.config, 158

d

docutils.parsers.rst, 161
sphinx.domains, 163

e

sphinx.environment, 159
sphinx.errors, 158
sphinx.ext.autodoc, 113
sphinx.ext.autosummary, 120
sphinx.ext.coverage, 133
sphinx.ext.doctest, 123
sphinx.ext.extlinks, 134
sphinx.ext.graphviz, 130
sphinx.ext.ifconfig, 133
sphinx.ext.inheritance_diagram, 132
sphinx.ext.intersphinx, 126
sphinx.ext.jsmath, 130
sphinx.ext.linkcode, 135
sphinx.ext.mathbase, 127
sphinx.ext.mathjax, 130
sphinx.ext.napoleon, 136
sphinx.ext.pngmath, 129
sphinx.ext.todo, 134
sphinx.ext.viewcode, 135

Symbols

–batchfile, –no-batchfile
sphinx-quickstart command line option, 10
–dot=DOT
sphinx-quickstart command line option, 9
–epub
sphinx-quickstart command line option, 10
–ext-autodoc
sphinx-quickstart command line option, 10
–ext-coverage
sphinx-quickstart command line option, 10
–ext-doctest
sphinx-quickstart command line option, 10
–ext-ifconfig
sphinx-quickstart command line option, 10
–ext-intersphinx
sphinx-quickstart command line option, 10
–ext-mathjax
sphinx-quickstart command line option, 10
–ext-pngmath
sphinx-quickstart command line option, 10
–ext-todo
sphinx-quickstart command line option, 10
–ext-viewcode
sphinx-quickstart command line option, 10
–makefile, –no-makefile
sphinx-quickstart command line option, 10
–master=MASTER
sphinx-quickstart command line option, 10
–sep
sphinx-quickstart command line option, 9
–suffix=SUFFIX
sphinx-quickstart command line option, 10
–use-make-mode, –no-use-make-mode
sphinx-quickstart command line option, 10
-A author
sphinx-apidoc command line option, 16
-A name=value
sphinx-build command line option, 12
-C
sphinx-build command line option, 12
-D setting=value
sphinx-build command line option, 12
-E
sphinx-build command line option, 12
-F, –full
sphinx-apidoc command line option, 15
-H project
sphinx-apidoc command line option, 16
-M
sphinx-apidoc command line option, 16
-N
sphinx-build command line option, 12
-P
sphinx-build command line option, 13
-Q
sphinx-build command line option, 13
-R release
sphinx-apidoc command line option, 16
-T
sphinx-build command line option, 13
-T, –no-toc
sphinx-apidoc command line option, 15
-V version
sphinx-apidoc command line option, 16
-W
sphinx-build command line option, 13
-a
sphinx-build command line option, 11
-a AUTHOR, –author=AUTHOR
sphinx-quickstart command line option, 9
-b buildername
sphinx-build command line option, 11
-c path
sphinx-build command line option, 12
-d maxdepth
sphinx-apidoc command line option, 15
-d path
sphinx-build command line option, 12
-f, –force
sphinx-apidoc command line option, 15

-h, –help, –version
sphinx-build command line option, 13
sphinx-quickstart command line option, 9
-j N
sphinx-build command line option, 12
-l LANGUAGE, –language=LANGUAGE
sphinx-quickstart command line option, 9
-l, –follow-links
sphinx-apidoc command line option, 15
-n
sphinx-build command line option, 12
-n, –dry-run
sphinx-apidoc command line option, 15
-o outputdir
sphinx-apidoc command line option, 15
-p PROJECT, –project=PROJECT
sphinx-quickstart command line option, 9
-q
sphinx-build command line option, 13
-q, –quiet
sphinx-quickstart command line option, 9
-r RELEASE, –release=RELEASE
sphinx-quickstart command line option, 9
-s suffix
sphinx-apidoc command line option, 15
-t tag
sphinx-build command line option, 12
-v
sphinx-build command line option, 12
-v VERSION
sphinx-quickstart command line option, 9
-w file
sphinx-build command line option, 13
$.getJSON() ($ method), 60

A

abbr (role), 40
abbreviation (class in sphinx.addnodes), 166
accept_comment() (sphinx.websupport.storage.StorageBackend method), 179
add_autodoc_attrgetter() (sphinx.application.Sphinx method), 153
add_autodocumenter() (sphinx.application.Sphinx method), 153
add_builder() (sphinx.application.Sphinx method), 150
add_comment() (sphinx.websupport.storage.StorageBackend method), 178
add_comment() (sphinx.websupport.WebSupport method), 175
add_config_value() (sphinx.application.Sphinx method), 150
add_crossref_type() (sphinx.application.Sphinx method), 152
add_directive() (sphinx.application.Sphinx method), 151
add_directive_to_domain() (sphinx.application.Sphinx method), 151
add_document() (sphinx.websupport.search.BaseSearch method), 177
add_domain() (sphinx.application.Sphinx method), 150
add_event() (sphinx.application.Sphinx method), 150
add_function_parentheses configuration value, 73
add_generic_role() (sphinx.application.Sphinx method), 152
add_index_to_domain() (sphinx.application.Sphinx method), 150
add_javascript() (sphinx.application.Sphinx method), 153
add_latex_package() (sphinx.application.Sphinx method), 153
add_lexer() (sphinx.application.Sphinx method), 153
add_module_names configuration value, 73
add_node() (sphinx.application.Sphinx method), 150
add_node() (sphinx.websupport.storage.StorageBackend method), 178
add_object_type() (sphinx.application.Sphinx method), 152
add_role() (sphinx.application.Sphinx method), 151
add_role_to_domain() (sphinx.application.Sphinx method), 151
add_search_language() (sphinx.application.Sphinx method), 154
add_stylesheet() (sphinx.application.Sphinx method), 153
add_transform() (sphinx.application.Sphinx method), 153
any (role), 37
app (sphinx.environment.BuildEnvironment attribute), 159
applehelp_bundle_id configuration value, 82
applehelp_bundle_name configuration value, 82
applehelp_bundle_version configuration value, 82
applehelp_codesign_flags configuration value, 83
applehelp_codesign_identity

configuration value, 83
applehelp_codesign_path
configuration value, 83
applehelp_dev_region
configuration value, 82
applehelp_disable_external_tools
configuration value, 83
applehelp_icon
configuration value, 82
applehelp_index_anchors
configuration value, 82
applehelp_indexer_path
configuration value, 83
applehelp_kb_product
configuration value, 82
applehelp_kb_url
configuration value, 82
applehelp_locale
configuration value, 83
applehelp_min_term_length
configuration value, 82
applehelp_remote_url
configuration value, 82
applehelp_stopwords
configuration value, 82
applehelp_title
configuration value, 83
AppleHelpBuilder (class in sphinx.builders.applehelp), 64
arguments (docutils.parsers.rst.Directive attribute), 161
attributes (built-in variable), 122
autoattribute (directive), 116
autoclass (directive), 114
autoclass_content
configuration value, 117
autodata (directive), 116
autodoc-process-docstring
event, 118
autodoc-process-signature
event, 118
autodoc-skip-member
event, 119
autodoc_default_flags
configuration value, 117
autodoc_docstring_signature
configuration value, 118
autodoc_member_order
configuration value, 117
autodoc_mock_imports
configuration value, 118
autoexception (directive), 114
autofunction (directive), 116
automatic
documentation, 113
linking, 126
testing, 123
automethod (directive), 116
automodule (directive), 114
autosummary (directive), 120
autosummary_generate
configuration value, 121

B

bar (directive), 61
BaseSearch (class in sphinx.websupport.search), 176
between() (in module sphinx.ext.autodoc), 119
block_text (docutils.parsers.rst.Directive attribute), 162
build() (sphinx.builders.Builder method), 160
build() (sphinx.websupport.WebSupport method), 173
build-finished
event, 157
build_all() (sphinx.builders.Builder method), 160
build_specific() (sphinx.builders.Builder method), 160
build_update() (sphinx.builders.Builder method), 160
BuildEnvironment (class in sphinx.environment), 159
builder, **185**
builder (built-in variable), 110
Builder (class in sphinx.builders), 160
builder-inited
event, 155

C

c:data (role), 54
c:func (role), 54
c:function (directive), 53
c:macro (directive), 53
c:macro (role), 54
c:member (directive), 53
c:type (directive), 53
c:type (role), 54
c:var (directive), 53
category (sphinx.errors.SphinxError attribute), 158
centered (directive), 31
changes
in version, 30
ChangesBuilder (class in sphinx.builders.changes), 66
CheckExternalLinksBuilder (class in sphinx.builders.linkcheck), 67
class (built-in variable), 122
classes (built-in variable), 122
clear_doc() (sphinx.domains.Domain method), 163

code
    examples, 33
code-block (directive), 34
codeauthor (directive), 43
command (role), 40
compact_paragraph (class in sphinx.addnodes), 166
compile() (built-in function), 51
conf (module), 69
Config (class in sphinx.config), 158
config (sphinx.environment.BuildEnvironment attribute), 159
ConfigError, 158
configuration directory, **185**
configuration value
    add_function_parentheses, 73
    add_module_names, 73
    applehelp_bundle_id, 82
    applehelp_bundle_name, 82
    applehelp_bundle_version, 82
    applehelp_codesign_flags, 83
    applehelp_codesign_identity, 83
    applehelp_codesign_path, 83
    applehelp_dev_region, 82
    applehelp_disable_external_tools, 83
    applehelp_icon, 82
    applehelp_index_anchors, 82
    applehelp_indexer_path, 83
    applehelp_kb_product, 82
    applehelp_kb_url, 82
    applehelp_locale, 83
    applehelp_min_term_length, 82
    applehelp_remote_url, 82
    applehelp_stopwords, 82
    applehelp_title, 83
    autoclass_content, 117
    autodoc_default_flags, 117
    autodoc_docstring_signature, 118
    autodoc_member_order, 117
    autodoc_mock_imports, 118
    autosummary_generate, 121
    copyright, 72
    coverage_c_path, 133
    coverage_c_regexes, 133
    coverage_ignore_c_items, 133
    coverage_ignore_classes, 133
    coverage_ignore_functions, 133
    coverage_ignore_modules, 133
    coverage_skip_undoc_in_source, 134
    coverage_write_headline, 133
    default_role, 71
    doctest_global_cleanup, 125
    doctest_global_setup, 125
    doctest_path, 125
    doctest_test_doctest_blocks, 125
    epub_author, 84
    epub_basename, 83
    epub_copyright, 84
    epub_cover, 84
    epub_exclude_files, 85
    epub_fix_images, 85
    epub_guide, 84
    epub_identifier, 84
    epub_language, 84
    epub_max_image_width, 85
    epub_post_files, 85
    epub_pre_files, 85
    epub_publisher, 84
    epub_scheme, 84
    epub_show_urls, 85
    epub_theme, 83
    epub_theme_options, 84
    epub_title, 84
    epub_tocdepth, 85
    epub_tocdup, 85
    epub_tocscope, 85
    epub_uid, 84
    epub_use_index, 86
    exclude_patterns, 70
    extensions, 69
    extlinks, 134
    gettext_additional_targets, 76
    gettext_auto_build, 75
    gettext_compact, 75
    gettext_location, 75
    gettext_uuid, 75
    graphviz_dot, 131
    graphviz_dot_args, 132
    graphviz_output_format, 132
    highlight_language, 73
    highlight_options, 73
    html_add_permalinks, 77
    html_additional_pages, 78
    html_compact_lists, 80
    html_context, 77
    html_copy_source, 79
    html_domain_indices, 78
    html_extra_path, 77
    html_favicon, 77
    html_file_suffix, 79
    html_last_updated_fmt, 77
    html_link_suffix, 79
    html_logo, 77
    html_output_encoding, 80
    html_scaled_image_link, 81
    html_search_language, 80
    html_search_options, 81
    html_search_scorer, 81
    html_secnumber_suffix, 80

html_short_title, 76
html_show_copyright, 80
html_show_sourcelink, 79
html_show_sphinx, 80
html_sidebars, 78
html_split_index, 79
html_static_path, 77
html_style, 76
html_theme, 76
html_theme_options, 76
html_theme_path, 76
html_title, 76
html_translator_class, 79
html_use_index, 79
html_use_modindex, 79
html_use_opensearch, 79
html_use_smartypants, 77
htmlhelp_basename, 81
inheritance_edge_attrs, 133
inheritance_graph_attrs, 132
inheritance_node_attrs, 132
intersphinx_cache_limit, 127
intersphinx_mapping, 126
jsmath_path, 130
keep_warnings, 71
language, 74
latex_additional_files, 88
latex_appendices, 87
latex_docclass, 88
latex_documents, 86
latex_domain_indices, 87
latex_elements, 87
latex_font_size, 89
latex_logo, 86
latex_paper_size, 89
latex_preamble, 89
latex_show_pagerefs, 87
latex_show_urls, 87
latex_use_modindex, 87
latex_use_parts, 86
linkcheck_anchors, 91
linkcheck_ignore, 91
linkcheck_timeout, 91
linkcheck_workers, 91
linkcode_resolve, 135
locale_dirs, 75
man_pages, 89
man_show_urls, 90
master_doc, 70
mathjax_path, 130
modindex_common_prefix, 73
napoleon_google_docstring, 139
napoleon_include_private_with_doc, 139
napoleon_include_special_with_doc, 140
napoleon_numpy_docstring, 139
napoleon_use_admonition_for_examples, 140
napoleon_use_admonition_for_notes, 140
napoleon_use_admonition_for_references, 141
napoleon_use_ivar, 141
napoleon_use_param, 141
napoleon_use_rtype, 142
needs_extensions, 71
needs_sphinx, 71
nitpick_ignore, 72
nitpicky, 72
numfig, 72
numfig_format, 72
numfig_secnum_depth, 72
pngmath_add_tooltips, 130
pngmath_dvipng, 129
pngmath_dvipng_args, 129
pngmath_latex, 129
pngmath_latex_args, 129
pngmath_latex_preamble, 129
pngmath_use_preview, 129
primary_domain, 71
project, 72
pygments_style, 73
release, 72
rst_epilog, 71
rst_prolog, 71
show_authors, 73
source_encoding, 70
source_parsers, 70
source_suffix, 70
template_bridge, 71
templates_path, 70
texinfo_appendices, 90
texinfo_documents, 90
texinfo_domain_indices, 90
texinfo_elements, 91
texinfo_no_detailmenu, 91
texinfo_show_urls, 90
text_newlines, 89
text_sectionchars, 89
today, 72
today_fmt, 72
todo_include_todos, 134
trim_doctest_flags, 73
trim_footnote_reference_space, 73
version, 72
viewcode_import, 135
xml_pretty, 92
connect() (sphinx.application.Sphinx method), 154
content (docutils.parsers.rst.Directive attribute), 161
content_offset (docutils.parsers.rst.Directive attribute), 162
contents

table of, 27
copyright
configuration value, 72
copyright (built-in variable), 110
coverage_c_path
configuration value, 133
coverage_c_regexes
configuration value, 133
coverage_ignore_c_items
configuration value, 133
coverage_ignore_classes
configuration value, 133
coverage_ignore_functions
configuration value, 133
coverage_ignore_modules
configuration value, 133
coverage_skip_undoc_in_source
configuration value, 134
coverage_write_headline
configuration value, 133
CoverageBuilder (class in sphinx.ext.coverage), 133
cpp:any (role), 58
cpp:class (directive), 54
cpp:class (role), 58
cpp:enum (directive), 56
cpp:enum (role), 58
cpp:enum-class (directive), 56
cpp:enum-struct (directive), 56
cpp:enumerator (directive), 56
cpp:enumerator (role), 58
cpp:func (role), 58
cpp:function (directive), 54
cpp:member (directive), 55
cpp:member (role), 58
cpp:namespace (directive), 56
cpp:namespace-pop (directive), 57
cpp:namespace-push (directive), 57
cpp:type (directive), 55
cpp:type (role), 58
cpp:var (directive), 55
cpp:var (role), 58
css_files (built-in variable), 109
cut_lines() (in module sphinx.ext.autodoc), 119

D

dangling_warnings (sphinx.domains.Domain attribute), 164
data_version (sphinx.domains.Domain attribute), 164
debug() (sphinx.application.Sphinx method), 155
debug2() (sphinx.application.Sphinx method), 155
default
domain, 71
role, 71
default-domain (directive), 48
default_role
configuration value, 71
delete_comment() (sphinx.websupport.storage.StorageBackend method), 178
deprecated (directive), 31
desc (class in sphinx.addnodes), 165
desc_addname (class in sphinx.addnodes), 165
desc_annotation (class in sphinx.addnodes), 166
desc_content (class in sphinx.addnodes), 166
desc_name (class in sphinx.addnodes), 166
desc_optional (class in sphinx.addnodes), 166
desc_parameter (class in sphinx.addnodes), 166
desc_parameterlist (class in sphinx.addnodes), 166
desc_returns (class in sphinx.addnodes), 166
desc_signature (class in sphinx.addnodes), 165
desc_type (class in sphinx.addnodes), 165
describe (directive), 59
DevhelpBuilder (class in sphinx.builders.devhelp), 64
dfn (role), 40
digraph (directive), 131
directive, **185**
Directive (class in docutils.parsers.rst), 161
directive() (sphinx.domains.Domain method), 163
directives (sphinx.domains.Domain attribute), 164
DirectoryHTMLBuilder (class in sphinx.builders.html), 63
disconnect() (sphinx.application.Sphinx method), 154
doc (role), 39
doc2path() (sphinx.environment.BuildEnvironment method), 159
docname (sphinx.environment.BuildEnvironment attribute), 159
docstitle (built-in variable), 110
docstring, 113
doctest, 123
doctest (directive), 123
doctest_global_cleanup
configuration value, 125
doctest_global_setup
configuration value, 125
doctest_path
configuration value, 125
doctest_test_doctest_blocks
configuration value, 125
doctree-read
event, 156
doctree-resolved
event, 156
doctreedir (sphinx.environment.BuildEnvironment attribute), 159
document name, **185**

documentation
automatic, 113
docutils.parsers.rst (module), 161
domain, **185**
Domain (class in sphinx.domains), 163
download (role), 39
download_reference (class in sphinx.addnodes), 167

E

embedded (built-in variable), 110
emit() (sphinx.application.Sphinx method), 154
emit_firstresult() (sphinx.application.Sphinx method), 154
enumerate() (built-in function), 5
env-before-read-docs
event, 155
env-get-outdated
event, 155
env-merge-info
event, 156
env-purge-doc
event, 155
env-updated
event, 157
environment, **185**
envvar (directive), 59
envvar (role), 40
epub_author
configuration value, 84
epub_basename
configuration value, 83
epub_copyright
configuration value, 84
epub_cover
configuration value, 84
epub_exclude_files
configuration value, 85
epub_fix_images
configuration value, 85
epub_guide
configuration value, 84
epub_identifier
configuration value, 84
epub_language
configuration value, 84
epub_max_image_width
configuration value, 85
epub_post_files
configuration value, 85
epub_pre_files
configuration value, 85
epub_publisher
configuration value, 84
epub_scheme
configuration value, 84
epub_show_urls
configuration value, 85
epub_theme
configuration value, 83
epub_theme_options
configuration value, 84
epub_title
configuration value, 84
epub_tocdepth
configuration value, 85
epub_tocdup
configuration value, 85
epub_tocscope
configuration value, 85
epub_uid
configuration value, 84
epub_use_index
configuration value, 86
EpubBuilder (class in sphinx.builders.epub), 64
eq (role), 128
event
autodoc-process-docstring, 118
autodoc-process-signature, 118
autodoc-skip-member, 119
build-finished, 157
builder-inited, 155
doctree-read, 156
doctree-resolved, 156
env-before-read-docs, 155
env-get-outdated, 155
env-merge-info, 156
env-purge-doc, 155
env-updated, 157
html-collect-pages, 157
html-page-context, 157
missing-reference, 156
source-read, 156
examples
code, 33
exceptions (built-in variable), 122
exclude_patterns
configuration value, 70
ExtensionError, 154, 158
extensions
configuration value, 69
extlinks
configuration value, 134
extract_context() (sphinx.websupport.search.BaseSearch method), 177

F

favicon (built-in variable), 110

feed() (sphinx.websupport.search.BaseSearch method), 177
file (role), 40
file_suffix (built-in variable), 110
final_argument_whitespace (docutils.parsers.rst.Directive attribute), 161
finish() (sphinx.builders.Builder method), 160
finish_indexing() (sphinx.websupport.search.BaseSearch method), 177
foo (directive), 61
foo (role), 61
found_docs (sphinx.environment.BuildEnvironment attribute), 159
fullname (built-in variable), 122
functions (built-in variable), 122

G

generate() (sphinx.domains.Index method), 165
get_data() (sphinx.websupport.storage.StorageBackend method), 179
get_data() (sphinx.websupport.WebSupport method), 174
get_document() (sphinx.websupport.WebSupport method), 174
get_objects() (sphinx.domains.Domain method), 163
get_outdated_docs() (sphinx.builders.Builder method), 160
get_relative_uri() (sphinx.builders.Builder method), 160
get_search_results() (sphinx.websupport.WebSupport method), 176
get_target_uri() (sphinx.builders.Builder method), 160
get_type_name() (sphinx.domains.Domain method), 163
gettext_additional_targets
    configuration value, 76
gettext_auto_build
    configuration value, 75
gettext_compact
    configuration value, 75
gettext_location
    configuration value, 75
gettext_uuid
    configuration value, 75
global
    substitutions, 71
globalcontext_filename (sphinx.builders.html.SerializingHTMLBuilder attribute), 66
glossary (class in sphinx.addnodes), 167
glossary (directive), 32
graph (directive), 131
graphviz (directive), 130
graphviz_dot
    configuration value, 131
graphviz_dot_args
    configuration value, 132
graphviz_output_format
    configuration value, 132
guilabel (role), 40

H

handle_query() (sphinx.websupport.search.BaseSearch method), 177
has_content (docutils.parsers.rst.Directive attribute), 161
has_source (built-in variable), 110
hasdoc() (built-in function), 110
highlight (directive), 34
highlight_language
    configuration value, 73
highlight_options
    configuration value, 73
highlightlang (class in sphinx.addnodes), 167
hlist (directive), 31
html-collect-pages
    event, 157
html-page-context
    event, 157
html_add_permalinks
    configuration value, 77
html_additional_pages
    configuration value, 78
html_compact_lists
    configuration value, 80
html_context
    configuration value, 77
html_copy_source
    configuration value, 79
html_domain_indices
    configuration value, 78
html_extra_path
    configuration value, 77
html_favicon
    configuration value, 77
html_file_suffix
    configuration value, 79
html_last_updated_fmt
    configuration value, 77
html_link_suffix
    configuration value, 79
html_logo
    configuration value, 77
html_output_encoding
    configuration value, 80
html_scaled_image_link
    configuration value, 81

html_search_language
configuration value, 80
html_search_options
configuration value, 81
html_search_scorer
configuration value, 81
html_secnumber_suffix
configuration value, 80
html_short_title
configuration value, 76
html_show_copyright
configuration value, 80
html_show_sourcelink
configuration value, 79
html_show_sphinx
configuration value, 80
html_sidebars
configuration value, 78
html_split_index
configuration value, 79
html_static_path
configuration value, 77
html_style
configuration value, 76
html_theme
configuration value, 76
html_theme_options
configuration value, 76
html_theme_path
configuration value, 76
html_title
configuration value, 76
html_translator_class
configuration value, 79
html_use_index
configuration value, 79
html_use_modindex
configuration value, 79
html_use_opensearch
configuration value, 79
html_use_smartypants
configuration value, 77
htmlhelp_basename
configuration value, 81
HTMLHelpBuilder (class in sphinx.builders.htmlhelp), 63

I

ifconfig (directive), 133
implementation (sphinx.builders.html.SerializingHTMLBuilder attribute), 65
in version
changes, 30
index (class in sphinx.addnodes), 166
Index (class in sphinx.domains), 164
index (directive), 43
index (role), 44
indices (sphinx.domains.Domain attribute), 164
info() (sphinx.application.Sphinx method), 154
inheritance-diagram (directive), 132
inheritance_edge_attrs
configuration value, 133
inheritance_graph_attrs
configuration value, 132
inheritance_node_attrs
configuration value, 132
init() (sphinx.application.TemplateBridge method), 158
init() (sphinx.builders.Builder method), 160
init_indexing() (sphinx.websupport.search.BaseSearch method), 176
initial_data (sphinx.domains.Domain attribute), 164
intersphinx_cache_limit
configuration value, 127
intersphinx_mapping
configuration value, 126

J

js:attr (role), 60
js:attribute (directive), 60
js:class (directive), 60
js:class (role), 60
js:data (directive), 60
js:data (role), 60
js:func (role), 60
js:function (directive), 59
jsmath_path
configuration value, 130
JSONHTMLBuilder (class in sphinx.builders.html), 66

K

kbd (role), 41
keep_warnings
configuration value, 71
keyword (role), 40

L

label (sphinx.domains.Domain attribute), 164
language
configuration value, 74
language (built-in variable), 110
last_updated (built-in variable), 110
latex_additional_files
configuration value, 88
latex_appendices
configuration value, 87
latex_docclass

configuration value, 88
latex_documents
configuration value, 86
latex_domain_indices
configuration value, 87
latex_elements
configuration value, 87
latex_font_size
configuration value, 89
latex_logo
configuration value, 86
latex_paper_size
configuration value, 89
latex_preamble
configuration value, 89
latex_show_pagerefs
configuration value, 87
latex_show_urls
configuration value, 87
latex_use_modindex
configuration value, 87
latex_use_parts
configuration value, 86
LaTeXBuilder (class in sphinx.builders.latex), 64
lineno (docutils.parsers.rst.Directive attribute), 162
linkcheck_anchors
configuration value, 91
linkcheck_ignore
configuration value, 91
linkcheck_timeout
configuration value, 91
linkcheck_workers
configuration value, 91
linkcode_resolve
configuration value, 135
linking
automatic, 126
literal_emphasis (class in sphinx.addnodes), 166
literalinclude (directive), 35
locale_dirs
configuration value, 75
logo (built-in variable), 110

M

mailheader (role), 41
makevar (role), 41
man_pages
configuration value, 89
man_show_urls
configuration value, 90
manpage (role), 41
ManualPageBuilder (class in sphinx.builders.manpage), 65
master document, **185**
master_doc
configuration value, 70
master_doc (built-in variable), 110
math (directive), 128
math (role), 128
mathjax_path
configuration value, 130
members (built-in variable), 122
menuselection (role), 41
merge_domaindata() (sphinx.domains.Domain method), 163
MessageCatalogBuilder (class in sphinx.builders.gettext), 66
meta (built-in variable), 111
meta (class in sphinx.addnodes), 167
metadata (sphinx.environment.BuildEnvironment attribute), 159
methods (built-in variable), 122
mimetype (role), 41
missing-reference
event, 156
modindex_common_prefix
configuration value, 73
module (built-in variable), 122
MyAnimal() (class), 60

N

name (built-in variable), 122
name (docutils.parsers.rst.Directive attribute), 161
name (sphinx.builders.applehelp.AppleHelpBuilder attribute), 64
name (sphinx.builders.changes.ChangesBuilder attribute), 67
name (sphinx.builders.devhelp.DevhelpBuilder attribute), 64
name (sphinx.builders.epub.EpubBuilder attribute), 64
name (sphinx.builders.gettext.MessageCatalogBuilder attribute), 66
name (sphinx.builders.html.DirectoryHTMLBuilder attribute), 63
name (sphinx.builders.html.JSONHTMLBuilder attribute), 66
name (sphinx.builders.html.PickleHTMLBuilder attribute), 66
name (sphinx.builders.html.SingleFileHTMLBuilder attribute), 63
name (sphinx.builders.html.StandaloneHTMLBuilder attribute), 63
name (sphinx.builders.htmlhelp.HTMLHelpBuilder attribute), 63
name (sphinx.builders.latex.LaTeXBuilder attribute), 64

name (sphinx.builders.linkcheck.CheckExternalLinksBuilder attribute), 67
name (sphinx.builders.manpage.ManualPageBuilder attribute), 65
name (sphinx.builders.qthelp.QtHelpBuilder attribute), 64
name (sphinx.builders.texinfo.TexinfoBuilder attribute), 65
name (sphinx.builders.text.TextBuilder attribute), 65
name (sphinx.builders.xml.PseudoXMLBuilder attribute), 67
name (sphinx.builders.xml.XMLBuilder attribute), 67
name (sphinx.domains.Domain attribute), 164
napoleon_google_docstring
    configuration value, 139
napoleon_include_private_with_doc
    configuration value, 139
napoleon_include_special_with_doc
    configuration value, 140
napoleon_numpy_docstring
    configuration value, 139
napoleon_use_admonition_for_examples
    configuration value, 140
napoleon_use_admonition_for_notes
    configuration value, 140
napoleon_use_admonition_for_references
    configuration value, 141
napoleon_use_ivar
    configuration value, 141
napoleon_use_param
    configuration value, 141
napoleon_use_rtype
    configuration value, 142
needs_extensions
    configuration value, 71
needs_sphinx
    configuration value, 71
new_serialno() (sphinx.environment.BuildEnvironment method), 159
newest_template_mtime() (sphinx.application.TemplateBridge method), 158
newsgroup (role), 41
next (built-in variable), 110
nitpick_ignore
    configuration value, 72
nitpicky
    configuration value, 72
note, 30
note (directive), 30
note_dependency() (sphinx.environment.BuildEnvironment method), 159
note_reread() (sphinx.environment.BuildEnvironment method), 160
numfig
    configuration value, 72
numfig_format
    configuration value, 72
numfig_secnum_depth
    configuration value, 72
numref (role), 39

O

object, **186**
object (directive), 59
object_types (sphinx.domains.Domain attribute), 164
objname (built-in variable), 122
ObjType (class in sphinx.domains), 164
only (class in sphinx.addnodes), 167
only (directive), 44
option (directive), 58
option (role), 40
option_spec (docutils.parsers.rst.Directive attribute), 161
optional_arguments (docutils.parsers.rst.Directive attribute), 161
options (docutils.parsers.rst.Directive attribute), 161
out_suffix (sphinx.builders.html.SerializingHTMLBuilder attribute), 66
override_domain() (sphinx.application.Sphinx method), 150

P

page_source_suffix (built-in variable), 112
pagename (built-in variable), 111
parents (built-in variable), 111
pathto() (built-in function), 109
pending_xref (class in sphinx.addnodes), 166
pep (role), 42
PickleHTMLBuilder (class in sphinx.builders.html), 66
pngmath_add_tooltips
    configuration value, 130
pngmath_dvipng
    configuration value, 129
pngmath_dvipng_args
    configuration value, 129
pngmath_latex
    configuration value, 129
pngmath_latex_args
    configuration value, 129
pngmath_latex_preamble
    configuration value, 129
pngmath_use_preview
    configuration value, 129

post_build() (sphinx.websupport.storage.StorageBackend method), 178
pre_build() (sphinx.websupport.storage.StorageBackend method), 178
prepare_writing() (sphinx.builders.Builder method), 160
prev (built-in variable), 111
primary
    domain, 71
primary_domain
    configuration value, 71
process_doc() (sphinx.domains.Domain method), 163
process_vote() (sphinx.websupport.storage.StorageBackend method), 179
process_vote() (sphinx.websupport.WebSupport method), 175
production (class in sphinx.addnodes), 167
productionlist (class in sphinx.addnodes), 167
productionlist (directive), 33
program (directive), 59
program (role), 41
project
    configuration value, 72
project (built-in variable), 111
PseudoXMLBuilder (class in sphinx.builders.xml), 67
py:attr (role), 52
py:attribute (directive), 50
py:class (directive), 49
py:class (role), 52
py:classmethod (directive), 50
py:const (role), 52
py:currentmodule (directive), 49
py:data (directive), 49
py:data (role), 52
py:decorator (directive), 50
py:decoratormethod (directive), 50
py:exc (role), 52
py:exception (directive), 49
py:func (role), 52
py:function (directive), 49
py:meth (role), 52
py:method (directive), 50
py:mod (role), 52
py:module (directive), 48
py:obj (role), 52
py:staticmethod (directive), 50
pygments_style
    configuration value, 73

Q

QtHelpBuilder (class in sphinx.builders.qthelp), 63
query() (sphinx.websupport.search.BaseSearch method), 177

R

ref (role), 38
regexp (role), 41
relbar() (built-in function), 110
reldelim1 (built-in variable), 109
reldelim2 (built-in variable), 109
release
    configuration value, 72
release (built-in variable), 111
relfn2path() (sphinx.environment.BuildEnvironment method), 159
rellinks (built-in variable), 111
render() (sphinx.application.TemplateBridge method), 158
render_string() (sphinx.application.TemplateBridge method), 158
require_sphinx() (sphinx.application.Sphinx method), 154
required_arguments (docutils.parsers.rst.Directive attribute), 161
resolve_any_xref() (sphinx.domains.Domain method), 163
resolve_xref() (sphinx.domains.Domain method), 164
rfc (role), 42
role, **186**
role() (sphinx.domains.Domain method), 164
roles (sphinx.domains.Domain attribute), 164
rst:dir (role), 61
rst:directive (directive), 60
rst:role (directive), 61
rst:role (role), 61
rst_epilog
    configuration value, 71
rst_prolog
    configuration value, 71
rubric (directive), 31
run() (docutils.parsers.rst.Directive method), 161

S

samp (role), 41
script_files (built-in variable), 109
searchindex_filename (sphinx.builders.html.SerializingHTMLBuilde attribute), 66
sectionauthor (directive), 43
seealso (class in sphinx.addnodes), 166
seealso (directive), 31
SerializingHTMLBuilder (class in sphinx.builders.html), 65
set_translator() (sphinx.application.Sphinx method), 150

setup_extension() (sphinx.application.Sphinx method), 150
shorttitle (built-in variable), 111
show_authors
  configuration value, 73
show_source (built-in variable), 111
sidebar() (built-in function), 110
SingleFileHTMLBuilder (class in sphinx.builders.html), 63
snippets
  testing, 123
source directory, **186**
source-read
  event, 156
source_encoding
  configuration value, 70
source_parsers
  configuration value, 70
source_suffix
  configuration value, 70
sourcecode, 33
sourcename (built-in variable), 111
Sphinx (class in sphinx.application), 149
sphinx-apidoc command line option
  -A author, 16
  -F, –full, 15
  -H project, 16
  -M, 16
  -R release, 16
  -T, –no-toc, 15
  -V version, 16
  -d maxdepth, 15
  -f, –force, 15
  -l, –follow-links, 15
  -n, –dry-run, 15
  -o outputdir, 15
  -s suffix, 15
sphinx-build command line option
  -A name=value, 12
  -C, 12
  -D setting=value, 12
  -E, 12
  -N, 12
  -P, 13
  -Q, 13
  -T, 13
  -W, 13
  -a, 11
  -b buildername, 11
  -c path, 12
  -d path, 12
  -h, –help, –version, 13
  -j N, 12
  -n, 12
  -q, 13
  -t tag, 12
  -v, 12
  -w file, 13
sphinx-quickstart command line option
  –batchfile, –no-batchfile, 10
  –dot=DOT, 9
  –epub, 10
  –ext-autodoc, 10
  –ext-coverage, 10
  –ext-doctest, 10
  –ext-ifconfig, 10
  –ext-intersphinx, 10
  –ext-mathjax, 10
  –ext-pngmath, 10
  –ext-todo, 10
  –ext-viewcode, 10
  –makefile, –no-makefile, 10
  –master=MASTER, 10
  –sep, 9
  –suffix=SUFFIX, 10
  –use-make-mode, –no-use-make-mode, 10
  -a AUTHOR, –author=AUTHOR, 9
  -h, –help, –version, 9
  -l LANGUAGE, –language=LANGUAGE, 9
  -p PROJECT, –project=PROJECT, 9
  -q, –quiet, 9
  -r RELEASE, –release=RELEASE, 9
  -v VERSION, 9
sphinx.addnodes (module), 165
sphinx.application (module), 149
sphinx.builders (module), 63
sphinx.builders.applehelp (module), 64
sphinx.builders.changes (module), 66
sphinx.builders.devhelp (module), 64
sphinx.builders.epub (module), 64
sphinx.builders.gettext (module), 66
sphinx.builders.html (module), 63
sphinx.builders.htmlhelp (module), 63
sphinx.builders.latex (module), 64
sphinx.builders.linkcheck (module), 67
sphinx.builders.manpage (module), 65
sphinx.builders.qthelp (module), 63
sphinx.builders.texinfo (module), 65
sphinx.builders.text (module), 65
sphinx.builders.xml (module), 67
sphinx.config (module), 158
sphinx.domains (module), 163
sphinx.environment (module), 159
sphinx.errors (module), 158
sphinx.ext.autodoc (module), 113
sphinx.ext.autosummary (module), 120
sphinx.ext.coverage (module), 133
sphinx.ext.doctest (module), 123

sphinx.ext.extlinks (module), 134
sphinx.ext.graphviz (module), 130
sphinx.ext.ifconfig (module), 133
sphinx.ext.inheritance_diagram (module), 132
sphinx.ext.intersphinx (module), 126
sphinx.ext.jsmath (module), 130
sphinx.ext.linkcode (module), 135
sphinx.ext.mathbase (module), 127
sphinx.ext.mathjax (module), 130
sphinx.ext.napoleon (module), 136
sphinx.ext.pngmath (module), 129
sphinx.ext.todo (module), 134
sphinx.ext.viewcode (module), 135
sphinx_version (built-in variable), 111
SphinxError, 158
src (docutils.parsers.rst.Directive attribute), 162
srcdir (sphinx.environment.BuildEnvironment attribute), 159
srcline (docutils.parsers.rst.Directive attribute), 162
StandaloneHTMLBuilder (class in sphinx.builders.html), 63
start_of_file (class in sphinx.addnodes), 167
state (docutils.parsers.rst.Directive attribute), 162
state_machine (docutils.parsers.rst.Directive attribute), 162
StorageBackend (class in sphinx.websupport.storage), 178
style (built-in variable), 111
substitutions
    global, 71
supported_image_types (sphinx.builders.applehelp.AppleHelpBuilder attribute), 64
supported_image_types (sphinx.builders.changes.ChangesBuilder attribute), 67
supported_image_types (sphinx.builders.devhelp.DevhelpBuilder attribute), 64
supported_image_types (sphinx.builders.epub.EpubBuilder attribute), 64
supported_image_types (sphinx.builders.gettext.MessageCatalogBuilder attribute), 66
supported_image_types (sphinx.builders.html.DirectoryHTMLBuilder attribute), 63
supported_image_types (sphinx.builders.html.JSONHTMLBuilder attribute), 66
supported_image_types (sphinx.builders.html.PickleHTMLBuilder attribute), 66
supported_image_types (sphinx.builders.html.SingleFileHTMLBuilder attribute), 63
supported_image_types (sphinx.builders.html.StandaloneHTMLBuilder attribute), 63
supported_image_types (sphinx.builders.htmlhelp.HTMLHelpBuilder attribute), 63
supported_image_types (sphinx.builders.latex.LaTeXBuilder attribute), 64
supported_image_types (sphinx.builders.linkcheck.CheckExternalLinksBuilder attribute), 67
supported_image_types (sphinx.builders.manpage.ManualPageBuilder attribute), 65
supported_image_types (sphinx.builders.qthelp.QtHelpBuilder attribute), 64
supported_image_types (sphinx.builders.texinfo.TexinfoBuilder attribute), 65
supported_image_types (sphinx.builders.text.TextBuilder attribute), 65
supported_image_types (sphinx.builders.xml.PseudoXMLBuilder attribute), 67
supported_image_types (sphinx.builders.xml.XMLBuilder attribute), 67

T

table of
    contents, 27
tabularcolumns (directive), 45
template_bridge
    configuration value, 71
TemplateBridge (class in sphinx.application), 158
templates_path
    configuration value, 70
term (role), 40
termsep (class in sphinx.addnodes), 167
testcleanup (directive), 123
testcode (directive), 124
testing
    automatic, 123
    snippets, 123
testoutput (directive), 124
testsetup (directive), 123
texinfo_appendices
    configuration value, 90

texinfo_documents
    configuration value, 90
texinfo_domain_indices
    configuration value, 90
texinfo_elements
    configuration value, 91
texinfo_no_detailmenu
    configuration value, 91
texinfo_show_urls
    configuration value, 90
TexinfoBuilder (class in sphinx.builders.texinfo), 65
text_newlines
    configuration value, 89
text_sectionchars
    configuration value, 89
TextBuilder (class in sphinx.builders.text), 65
ThemeError, 158
title (built-in variable), 111
titles (sphinx.environment.BuildEnvironment attribute), 159
toc (built-in variable), 111
toctree (built-in variable), 111
toctree (class in sphinx.addnodes), 167
toctree (directive), 27
today
    configuration value, 72
today_fmt
    configuration value, 72
todo (directive), 134
todo_include_todos
    configuration value, 134
todolist (directive), 134
token (role), 40
trim_doctest_flags
    configuration value, 73
trim_footnote_reference_space
    configuration value, 73

U

underline (built-in variable), 122
update_username() (sphinx.websupport.storage.StorageBackend method), 179
use_opensearch (built-in variable), 111

V

verbose() (sphinx.application.Sphinx method), 155
version
    configuration value, 72
version (built-in variable), 111
version_info (in module sphinx), 157
versionadded (directive), 30
versionchanged (directive), 31
versionmodified (class in sphinx.addnodes), 166
VersionRequirementError, 158
viewcode_import
    configuration value, 135

W

warn() (sphinx.application.Sphinx method), 154
warn() (sphinx.environment.BuildEnvironment method), 159
warn_node() (sphinx.environment.BuildEnvironment method), 159
warning, 30
warning (directive), 30
WebSupport (class in sphinx.websupport), 173
write_doc() (sphinx.builders.Builder method), 160

X

xml_pretty
    configuration value, 92
XMLBuilder (class in sphinx.builders.xml), 67